Digital Technology Incorporated

WWMCCS
Host to Front End Protocols: Specifications Version 1.0

John D. Day
Gary R. Grossman
Richard H. Howe

5 November 1979
DTI Document 78012.C-INFE.14

(2) LEVEL II

DTI ~~Document~~ -78012.C-INFE.14

WWMCCS
HOST TO FRONT END PROTOCOLS:
Specifications

by

John D. Day
Gary R. Grossman
Richard H. Howe

Prepared for the
Defense Communications Agency
under contract
DCA100-77-C-0069
Phase B Network Front End Research and Development

by

DIGITAL TECHNOLOGY INCORPORATED
302 E. John, Champaign, Illinois 61820

5 November 1979

Approved for Release: Jody S. Kravitz

Jody S. Kravitz, Project Manager

ABSTRACT

This document presents the specifications of the WWMCCS host to front end protocols. A brief overview of the WWMCCS network front end protocol architecture is presented. The link protocol is functionally specified. The channel protocol is completely specified. The complete channel protocol specification includes a narrative overview of the channel protocol mechanisms, a detailed treatment of each channel protocol message and event type, and a complete state table. A meta-specification for the service access protocols is presented.

Table of Contents

Table of Contents Continued

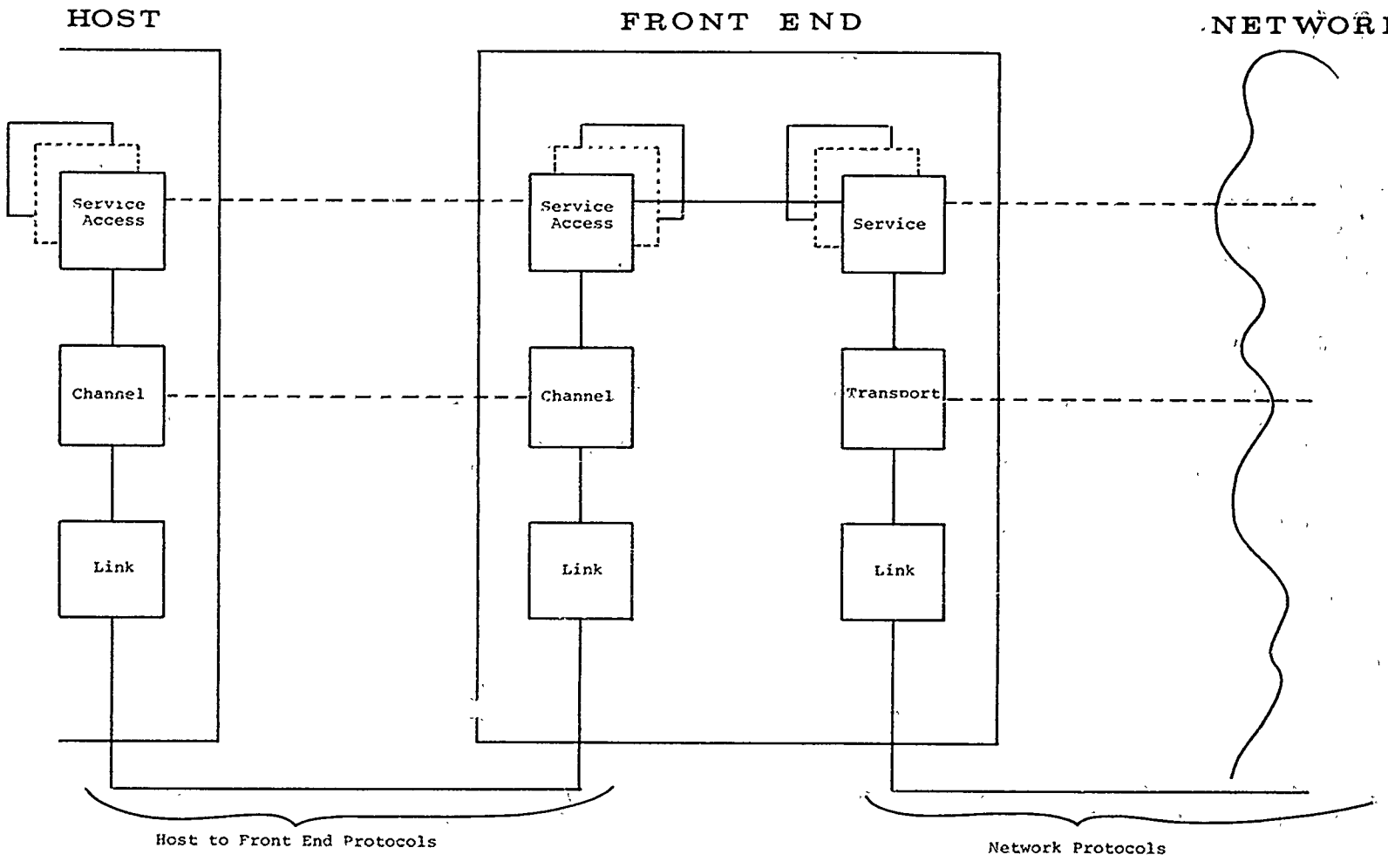


Figure 1: NETWORK FRONT END PROTOCOL ARCHITECTURE

NETWORK FRONT END PROTOCOL ARCHITECTURE

Network Front End

A network front end (NFE) is a computer system that interfaces a host computer and terminals to a network. A host connected directly to a network must support the often large and complex protocol interpreters for the network services. An NFE implements these protocol interpreters for the host. An NFE thus relieves the host of a considerable burden of protocol processing. A front ended host need support only the relatively simple interpreters for the host to front end protocols. This frees host resources for applications use. An NFE also provides terminal access to the network. This further frees host resources and also provides for continued access to the network in the event of host failure.

NFE Protocol Architecture

An NFE implements two different sets of protocols (see Figure 1):

1. the protocols for the network communication services, and
2. the host to front end protocols.

The protocols for the network services include:

1. a link protocol for communication between the front end and the local packet switch,
2. a transport protocol for reliable data transfer between local and remote subscriber processes, and

3. one or more higher order service protocols for, e.g., a network virtual terminal or a file transfer service.

The host to front end protocols include:

1. a link protocol for communication between the host and the front end,

2. a "channel" protocol for reliable data transfer between host and front end processes, and

3. for each network (or other) service supported by the front end, a "service access" protocol which provides for mutual access between host applications and the service.

The interpreters for these protocols have their apposites* in the host.

As Figure 1 indicates, both the network services and the host to front end protocols are layered. The network layers are, from bottom to top: the link layer, the transport layer, and the higher order service(s) layer(s). The host to front end layers are, from bottom to top: the link layer, the channel layer, and the service access layer.

* "Apposite": Having the same syntactic relation. E.g., in Figure 1, the channel protocol interpreter in the host and the channel protocol interpreter in the front end are each other's apposites.

NOTE ON THE SPECIFICATIONS

Specifying the Link Protocol

The link protocol definition depends on the link hardware. Thus, the link protocol cannot be specified without reference to a particular installation. However, the properties of the virtual communications medium that the link protocol implementation must provide have been functionally specified in the present document. When a link protocol satisfying this functional specification has been accepted for a given installation, the complete link protocol specification should be appended to the present document.

Specifying the Channel Protocol

The channel protocol definition is implementation independent. Thus, the channel protocol has been completely specified in the present document. Included in this specification is a descriptive model of the interface between a channel protocol interpreter (CPI) and any service access protocol interpreters (SAPIs) or other processes that communicate directly with a CPI. The interface between a CPI and the link protocol interpreter is to be defined by the link protocol. Therefore,

the CPI-link interface has not been specified in the present document.

Specifying the Service Access Protocols

The service access protocol definitions depend on the services to which they provide access. Thus, the service access protocols cannot be specified without reference to the particular services. For the same reason, they cannot be functionally specified. However, to ensure uniform documentation, a meta-specification of the service access protocols is presented in the present document. Whenever a service access protocol satisfying this meta-specification has been designed, the complete service access protocol specification should be appended to the present document.

Note: every SAPI communicates directly with its CPI. This direct communication is defined by the channel protocol (see Channel Interface). In some cases, processes other than SAPIs may communicate with one another using the channel protocol implementation as their communications medium. Such processes also communicate directly with their CPIs. Whenever such a process is to be implemented, the complete specification of its use of the channel interface and, if appropriate, its use of service access protocol messages (see Specifying Service Access Protocols) should be appended to the present document.

LINK PROTOCOL

Specification

The hardware link between the host and the front end may differ from site to site. Thus, the link protocol cannot be defined without reference to a given installation. For each installation, a link protocol must be implemented which provides efficient bi-directional communication, i.e., which efficiently uses the available host and front end resources as well as the communications hardware. In addition to efficiency, the link protocol implementation must provide the channel protocol interpreters (CPIs) with a virtual communications medium having the following properties:

1. the appearance of full duplex communication, independently of whether the link itself is full or half duplex,
2. delivery of data in-order and without duplication or loss,
3. flow control,
4. bit stream transparency,
5. fault notification, and
6. quality of service defined as
    a) an undetected bit error rate less than 10**-12,
    b) undetected data loss rate less than 10**-15, and
    c) undetected misdelivery rate less than 10**-15.

If the link protocol does not provide the CPIs with a virtual

communications medium satisfying these criteria without exception, their correct operation cannot be guaranteed.

When a link protocol satisfying these criteria has been accepted, the complete link protocol specification should be appended to the present document. Either an existing or a new link protocol may be used. If an existing link protocol is used, its specification should already include the following information:

1) a reference to the document describing the link protocol used,
2) the version of the protocol,
3) the options implemented,
4) the parameter values used, such as time-out values, etc.
5) a detailed description of any local conventions or modifications to the protocol.

If any of this information is missing, the specification should be augmented to include it.

If the link protocol accepted has been especially designed for a particular host-front end configuration, then a link protocol specification must be written. This specification should follow as closely as possible the method of specification used in the present document.

BLANK PAGE

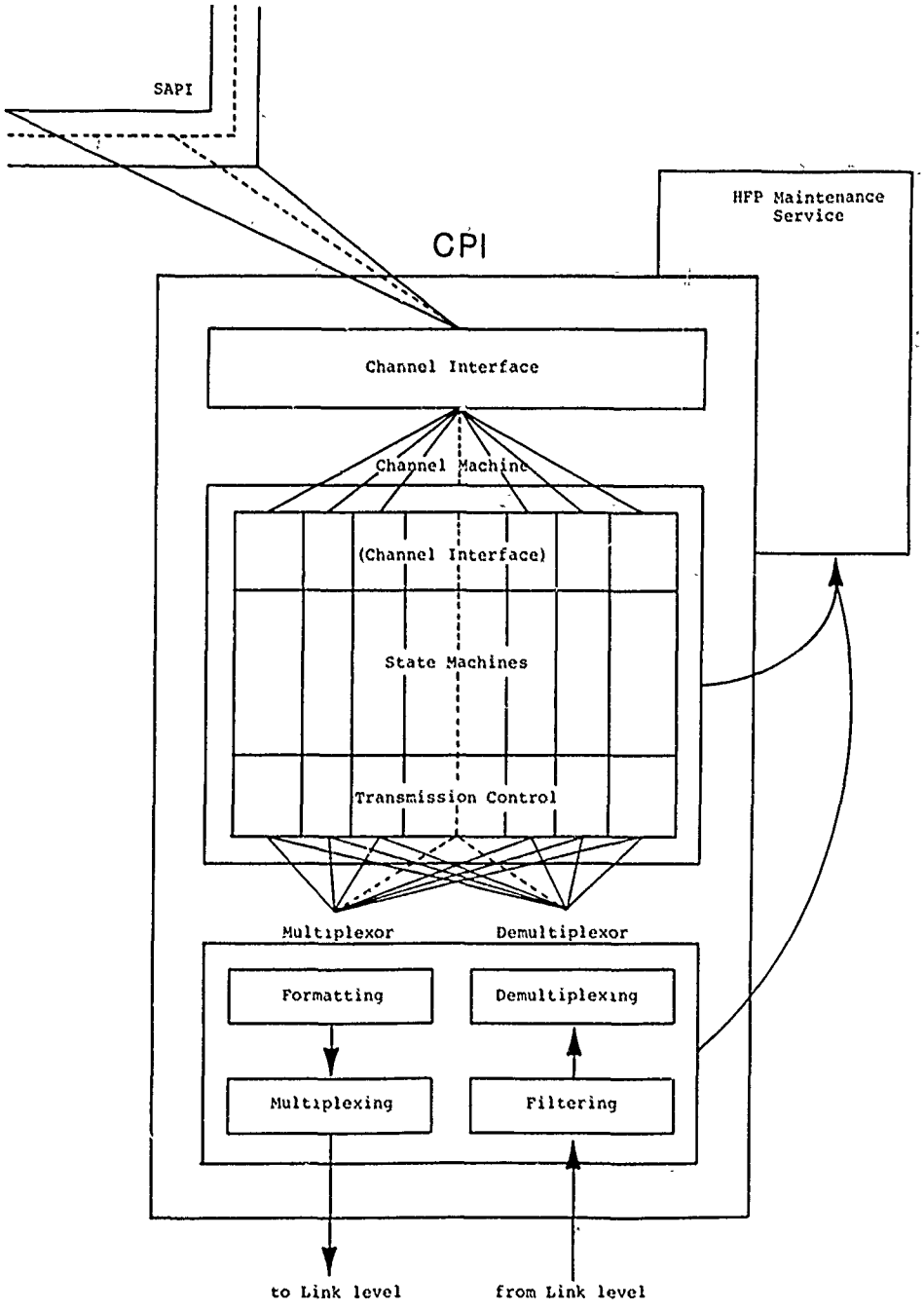


Figure 2: CHANNEL PROTOCOL INTERPRETER FUNCTIONS

CHANNEL PROTOCOL

Specification

Overview

The host to front end channel protocol defines the virtual communications medium for the service access protocol interpreters (SAPIs) (see Figure 1). This communications medium appears to the SAPIs as a set of host to front end "virtual channels", each of which will support a single service access connection. The channel protocol provides for establishing and terminating these channels and, together with the link protocol, provides for duplicate free, loss free, and error free data exchange, which may be either ordered with flow control or unordered without flow control.

The units of communication between channel protocol interpreters (CPIs) are called "channel protocol messages". The units of communication at the interface between a CPI and an SAPI are called "channel interface events". The channel protocol defines:

1) the channel messages and channel interface events,
2) their use in establishing and terminating channels and in effecting the flow of data over them, and
3) the states of each end of a channel together with the state transition rules.

Each CPI consists functionally of a channel interface, channel machines, and a multiplexor/demultiplexor (see Figure 2). The channel interface provides access to the channels for the SAPIs.

Each channel machine maintains the state of one end of a channel. The multiplexor/demultiplexor multiplexes messages coming from the channel machines and formats them for transmission via the link level. It filters messages coming from the link level and demultiplexes them to their respective channel machines.

Closely allied with each CPI is an HFP Maintenance Service process that participates in initializing host-front end communication, records errors reported by the CPI(s), and communicates these error reports between apposite CPIs.

Channel Protocol Messages

There are five types of channel protocol messages: Begin, Transmit, Execute, End, and Nop. A message of a given type may be either a Command or a Response.

Begin_Commands and Begin_Responses are used by CPIs to establish channels between them.

Transmit_Commands and Transmit_Responses are used by CPIs to effect data exchange over the channels. The use of Transmit_Commands and Transmit_Responses provides for maintaining order and flow control.

Execute_Commands and Execute_Responses are also used by CPIs to effect data exchange over the channels, but the use of Execute_Commands and Execute_Responses does not provide for maintaining order or flow control.

End_Commands and End_Responses are used by CPIs to terminate one or more of the channels between them.

Nops are used by CPIs as filler when channel protocol messages do not completely fill link protocol frames.

Channel Machines

A channel machine consists functionally of a state machine, a transmission controller, and a channel interface (see Figure 2).

The state machine maintains the state of its end of the channel. The state machine changes state in response to the messages and interface events received by the channel machine.

The transmission controller provides message flow control, duplicate message detection, and out of order message detection (see Transmission Control).

The channel interface receives channel interface events from the SAPI and the state machine, performs consistency checks on them, and passes them on to the state machine or the SAPI, respectively.

Channel Multiplexor/Demultiplexor

The channel multiplexor/demultiplexor performs multiplexing, demultiplexing, formatting, and filtering functions for the CPI. Messages generated by the CPI are formatted according to the channel protocol message formats. Messages from the channel machines, the filter, and the demultiplexor are multiplexed into a single stream and passed to the link level. The multiplexor ensures that each channel receives its share of the link level bandwidth.

Messages received from the link level are checked for correctness and consistency. Messages failing these checks are filtered out of the data stream. The appropriate Response (if any) is formulated and passed to the multiplexor for transmission to the apposite CPI. The HFP Maintenance Service is notified of the error. Messages passing these checks are sent to the demultiplexor.

The demultiplexor passes each message to the appropriate channel machine, if possible. If there is no channel machine to which to pass the message, the demultiplexor formulates the appropriate Response (if any), passes the Response to the multiplexor for transmission to the apposite CPI, and notifies the HFP Maintenance Service of the error.

Message Formatting and Filtering

This section specifies the format of channel protocol messages and the consistency checks to be performed on each message before accepting it.

Channel Protocol Message Formatting

In order to simplify decoding and buffer management, all channel protocol messages have the same basic format. This format is shown in Figure 3 (see p. 14).

Parts of Messages

Each message has three parts.

Part	Function
HEADER	comprises the fields containing the information for executing the channel protocol for this message (see below).
PAD	is zero or more bits long (an installation parameter) and serves only to place TEXT on a boundary convenient for both parties to the protocol.
TEXT	contains a service access or other higher level protocol message. Since the Size field (defined below) contains the number of bits in the entire message, and since the HEADER is 72 bits long, the size of TEXT is: (Size) - (size of PAD) - 72.

Message Format

Field Name	Field Size (bits)	Alternate Field Size (bits)	Alternate Field Name
HEADER			
Size	16		
Type	3		
C/R	1		
Credit	4		
Seq	4	8	Service (BEGIN Command)
Ack	4		
(not used)	4		
Group	12		
Member	16		
Control (Commands)	8	8	Status (Responses)
PAD	Note A		
TEXT	Note B		

Note A: The size of PAD is an installation parameter.
Note B: The size of TEXT is computed by:
(Size) - (size of PAD) - 72.

Figure 3: CHANNEL PROTOCOL MESSAGE FORMAT

HEADER Fields

The functions of the HEADER fields are defined below.

Field	Function
Size	specifies the number of bits in the entire message. This field is 16 bits long. It allows the representation of message sizes of up to 65,535 bits. Actual maximum message size is an installation parameter which may be less than this.
Type	specifies the message type: Begin Ø Transmit 1 Execute 3 End 4 Nop 5
C/R	specifies whether the message is a Command (C/R = Ø) or a Response (C/R = 1).
Credit	specifies the number of Transmit_Commands beyond the number specified by the Ack field (see below), which the sender of this message is prepared to receive.
Seq	specifies, in a Transmit_Command, its sequence number. Specifies, in all other messages (except Begin_Commands), the sequence number of the last Transmit_Command sent by the sender of this message. In Begin_Commands, both the Seq and Ack fields are replaced by the Service field.
Ack	specifies the sequence number of the last in-sequence Transmit_Command correctly received by the sender of this message (except in Begin_Commands). In Begin_Commands, both the Seq and Ack fields are replaced by the Service field.
Service	specifies, in Begin_Commands, the SAPI

to which the channel is to be established. The Service field occupies the same space as the Seq and Ack fields in all other types of messages.

Group — specifies the channel group which the message references (see Addressing Conventions).

Member — specifies the channel which the message references within the channel group (see Addressing Conventions).

Control — specifies control information for Execute_Commands and End_Commands. Its use in other Commands is currently undefined. The Control field in Commands occupies the same space as the Status field in Responses.

Status — specifies status information in Responses. The Status field in Responses occupies the same space as the Control field in Commands.

Channel Protocol Message Filtering

When a channel protocol message is received by a CPI, it must be checked for validity. Before the message is demultiplexed, the filter function of the CPI performs validity checks on it (see Figure 2). These validity checks are for:

Message Size: Compare message length for agreement with the contents of the Size field. It is assumed that the length of a message can be determined independently either from the hardware or from the link protocol interpreter.

Note: The hardware determined length may be greater than the contents of the Size field. (In some cases, the hardware will pad the message out to a word boundary on the receiving system.) In such a case, the validity check can only be for the hardware determined length being less than the contents of the Size field.

Message Type: check the contents of the Type field to determine whether or not the value represents a valid Type.

Message too large: check the Size of the message to determine whether or not it exceeds the maximum allowed at this site.

Group and Member fields: check the Group and Member fields to determine whether or not the message references a valid Group and Member.

Note: Since this check is tantamount to demultiplexing a message, it may be performed as part of that function.

Service: if the message is a Begin_Command, then check to determine whether or not a valid SAPI is being requested.

Control field: check the Control field (if defined) of each Command to determine whether or not the value is valid for this Command type.

Status field: check the Status field of each Response to determine whether or not the value is valid for this Response type.

If an error is detected in a message, the normal procedure is

to log the error in an error log, send a Response (if any) notifying the apposite CPI, and discard the message. There are two exceptions. First, if a message fails either the Size or Type field checks, the CPI cannot trust any of the information in the HEADER to be correct. Therefore, the CPI cannot determine what kind of Response to respond with. Second, if the CPI detects an error in a Response, it cannot send a Response to a Response. In either case, the CPI uses the HFP Maintenance Service to notify the apposite CPI of the error.

Note: since a CPI only communicates with only one other CPI (and not with many), it is not necessary that all of the above checks be made at all times. The checks for valid Service, Control, and Status may be made during a testing phase of operation and omitted during normal operation.

Status Codes

Every Response contains a Status field whose value indicates the success of or the reason for failure of the Command to which it is a Response. The Status field value conventions are:

a. zero indicates that the action initiated by the Command was successful;

b. 1 to 31 indicate errors applicable to all Commands at the channel protocol level;

c. 32 to 63 indicate errors specific to individual Command types at the Channel Protocol level;

d. 64 to 255 are reserved for internal use within the host and front end.

The codes common to all Responses are summarized below.

Status	Meaning
Ø	Command was successful.
1	Channel non-existent: the Group and Member fields of a Command (other than a Begin_Command) referenced a channel machine unknown to the receiving CPI.
2	Illegal state: a Command referenced a channel machine which was in a state for which the Command is an illegal input.
3	Command not implemented: a Command was received whose Type is legal but not implemented. Currently this can only be a Begin_Command or an Execute_Command.
5	Message too long: the number of bits in the Command exceeded the maximum permitted by the receiving CPI.
6	Service access protocol message error: an error in the service access protocol message

contained in the TEXT field of the Command was detected by the SAPI.

7 Illegal Control field value: the Control field of the Command contained an undefined value.

The Status codes specific to each Response are given under its specification (see Commands and Responses).

Message Multiplexing and Demultiplexing

Addressing Conventions

Each bi-directional service access connection is supported by a single host to front end channel. Usually, many such channels must be maintained over a single physical connection. A number is therefore assigned to each channel to identify it. It may be desirable to group channels (e.g., according to service) and manipulate the group as a whole. For this reason, the channel identifier is divided into two fields called Group and Member. A channel number whose Member field has the value zero refers to all channels of the group. The channel protocol defines the following conventions for these fields:

1) An End_Command with Group not equal to zero and Member equal to zero terminates all channels in the specified group.

2) An End_Command with Group equal to zero and Member equal to zero terminates all channels between the two apposite CPIs.

The Group and Member fields in a message HEADER are specified to be large enough (12 and 16 bits, respectively) to allow an installation to place all channels in a single group, to place each channel in its own group, or to use the full power of the two-dimensional channel address structure. A channel is assigned its number by the CPI which initiates its establishment. Since apposite CPIs refer to a particular

channel by the same channel number, and because either CPI may initiate channel establishment, name space conflicts must be avoided. This is accomplished by pre-assigning half the name space to each end. The high order bit of the Group field is used to distinguish the two ends. The host owns the half of the name space with the high order bit of the Group field equal to zero. The front end owns the half of the name space with the high order bit of the Group field equal to one.

Multiplexing

The unit of multiplexing is the channel protocol message. All messages from the channel machines and any that may be generated by the filtering and demultiplexing functions are passed to the multiplexor for multiplexing into the link level's single data stream.

Demultiplexing

Messages received via the link level from the apposite CPI are passed to the demultiplexor in accord with the state of flow control at the link level interface. The messages are then checked for correctness and consistency (see Message Formatting and Filtering). If a message passes these checks, the demultiplexor uses the Group and Member fields of the message HEADER to determine which channel machine(s) the

message addresses. The message is then passed to the indicated channel machines(s). If the message is a Begin_Command, a new channel machine will be created to process this request for a connection.

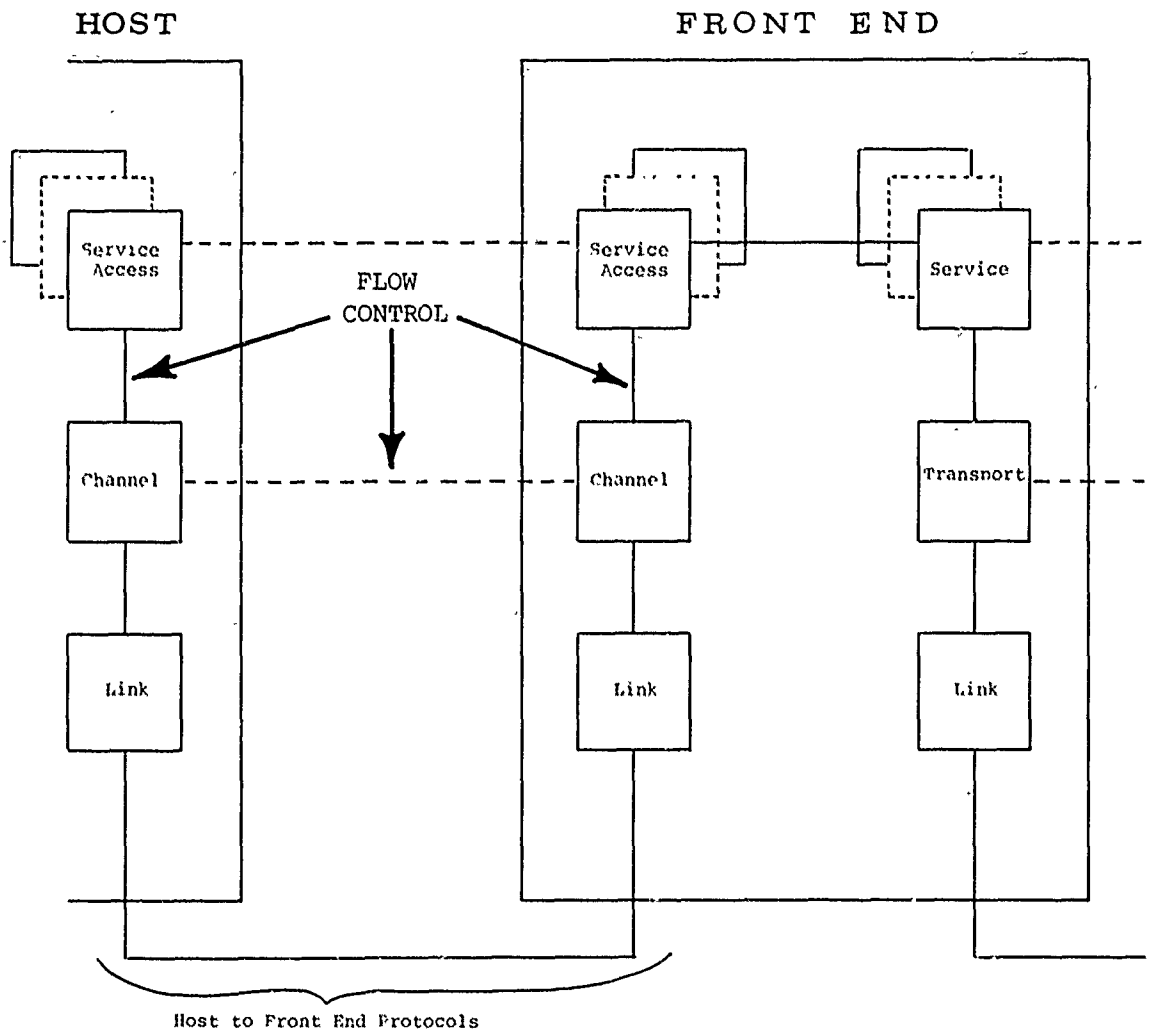


Figure 4: CHANNEL PROTOCOL FLOW CONTROL POINTS

Transmission Control

Together, the channel and link protocols provide for duplicate-free, loss-free, and error-free data exchange. The exchange of data between apposite CPIs may be either ordered with flow control ("controlled transmission"), or unordered without flow control ("uncontrolled transmission"). The link protocol provides for error detection, and lost and duplicate message detection. However, duplicate messages may be re-introduced by retransmissions at the channel level. Duplicates thus introduced are detected by the CPI and removed.

In addition to defining these functions for ensuring the integrity of the data stream, the channel protocol also defines the interaction between controlled and uncontrolled transmission, and the flow control mechanisms for the individual channels.

Controlled Transmission Mechanisms

The channel protocol implementation provides a controlled data stream to apposite SAPIs. The controlled data stream preserves order (i.e., data is delivered to the receiving SAPI in the same order in which it was presented by the sending SAPI) and is flow controlled. Flow control mechanisms are employed at three different points in the controlled data stream between apposite SAPIs (see Figure 4). This threefold flow control prevents a slow receiving SAPI from being overrun by a faster sending SAPI. The channel level flow control also

helps to ensure fair use of the link level by preventing any single channel machine from flooding the multiplexor with messages. The controlled transmission data stream is provided via the Transmit_Command and the Credit, Seq, and Ack fields of other Commands and Responses and via the s_ready and ci_ready channel interface events.

Credit: flow control between apposite channel machines is provided by the receiver of Transmit_Commands granting Credit to the sender. Credit can be sent in any channel protocol message (except End_Commands and End_Responses). The Credit field of a message contains the number of Transmit_Commands the sender may send beyond the last message acknowledged. In other words, the value of the Ack field added modulo 16 to the value of the Credit field is the largest sequence number the receiving channel machine is currently willing to accept. The flow control mechanisms are described in greater detail in the section on Flow Control below.

Sequence Numbers: order is preserved by assigning a unique sequence number to each Transmit_Command. Sequence numbers are assigned in ascending order modulo 16. The sequence numbers are the basis for ordering, duplicate detection, acknowledgement, and flow control. Each message type (except Begin_Commands) contains a Seq field. In messages other than Transmit_Commands, Seq specifies the sequence number of the last Transmit_Command sent by the sender of the message. In Begin_Commands the value of Seq must be zero.

Acks: delivery confirmation of Transmit_Commands is accomplished via acknowledgements. Each message type (except Begin_Commands) contains an Ack field. An Ack is the sequence number of the last Transmit_Command correctly received by the receiving channel machine. The Ack confirms the delivery of all messages with sequence numbers less than or equal to the sequence number in the Ack field. In Begin_Commands, the value of Ack must be zero. Arithmetic and comparisons on Acks is done modulo 16, and, in order to avoid ambiguous interpretation of sequence numbers, a channel machine cannot have more than eight unacknowledged messages outstanding.

Sending Queue: since the sending channel machine may send data to its apposite faster than it can be accepted, flow control is used to prevent the sender from flooding the receiver. Therefore the sending channel machine has a queue for messages waiting until flow control allows them to be sent. Messages from the controlled data stream are entered into this queue first in first out (FIFO). Messages from the uncontrolled data stream may or may not not be entered into this queue according to a FIFO discipline (see Uncontrolled Transmission).

Channel Interface Flow Control: a channel machine and an SAPI communicate via the channel interface. Flow control must be provided across this interface to prevent the SAPI from flooding the channel machine and vice-versa. In the model of the interface described in this specification, the s_ready and ci_ready events provide flow control across the channel interface. The s_ready event indicates to the channel machine the number of ci_transmit_c events the SAPI is able to accept. Similarly, the ci_ready event indicates to the SAPI the number of s_transmit_c events the channel machine is able to accept.

Receiving Queue: since the channel machine may send data to the SAPI faster than the SAPI can accept it, flow control is required across the channel interface. Therefore, the channel machine has a queue for events waiting until the channel interface flow control allows them to be sent to the SAPI. Events from the controlled data stream are entered into the queue first in first out (FIFO). Events from the uncontrolled data stream may or may not be entered into this queue according to a FIFO discipline. (See Uncontrolled Transmission).

BLANK PAGE

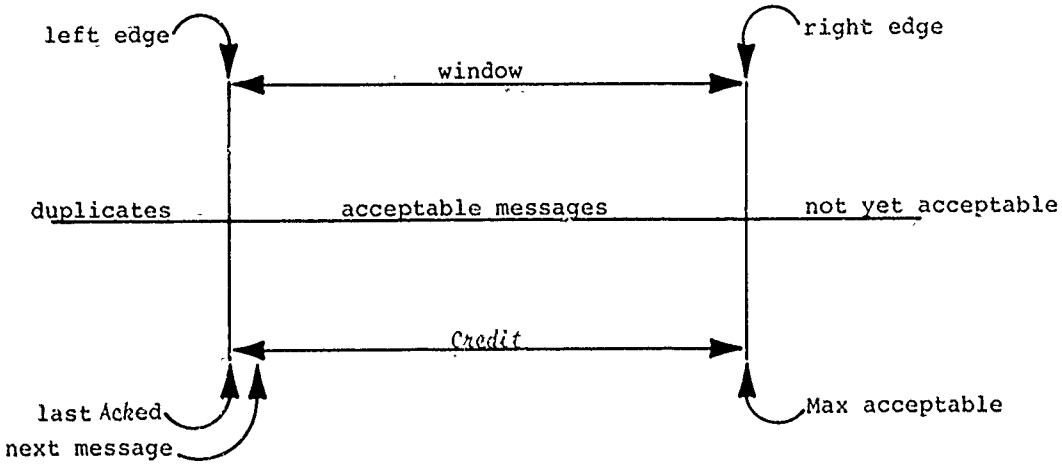


Figure 5a: THE MOVING WINDOW MODEL

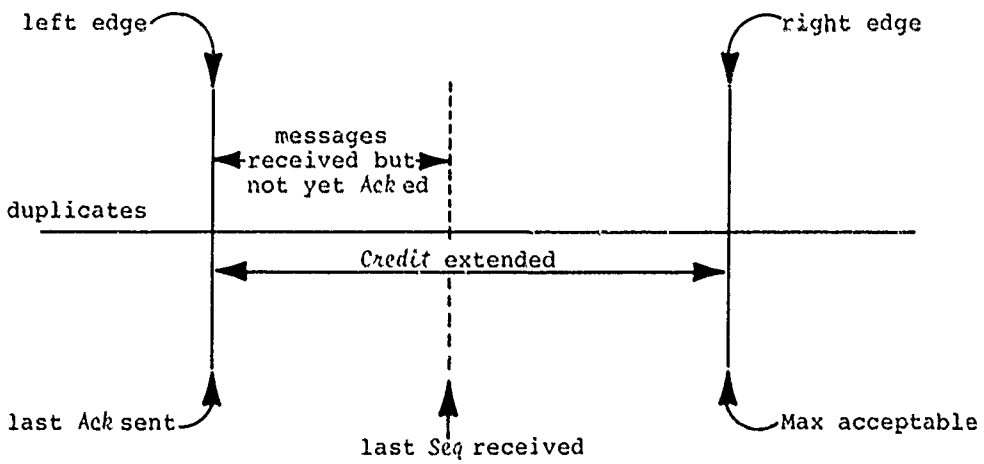


Figure 5b: s_window

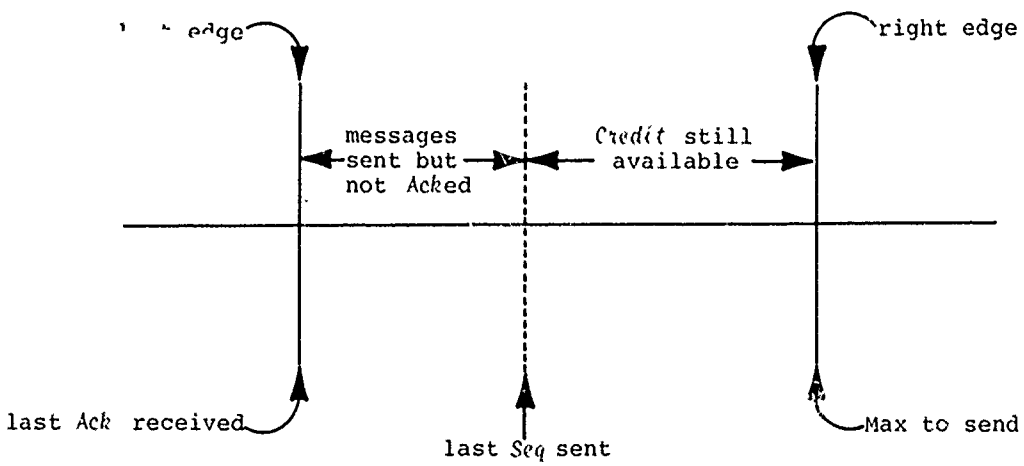


Figure 5c: r_window

Controlled Transmission Procedures

The Moving Window Model: the mechanisms used in the Channel Protocol for preserving message order, detecting duplicates, and controlling data flow are based on the moving window model. In this model, a window is seen to be sliding along the sequence number line (see Figure 5a). Only messages whose sequence numbers are within the window are acceptable. Sequence numbers less than the left edge of the window denote messages that have already been acknowledged. The left edge itself denotes the sequence number of the last in-order message received. The width of the window denotes the amount of Credit currently extended to the sender, i.e., the number of messages which the receiver is currently willing to accept. Sequence numbers greater than or equal to the right edge of the window denote messages that are not yet acceptable because they are beyond the receiver's current ability to accept them.

For each direction of data flow, there are two windows (see Figures 5b and 5c), one maintained by the sender (the "s_window") and one maintained by the receiver (the "r_window"). Data flow is controlled by the receiver. The receiver's r_window represents the receiver's image of the state of the data flow: the last message acknowledged by the receiver and the amount of Credit extended to the sender. The sender's s_window represents the sender's image of the receiver's r_window: the last acknowledged message and the amount of Credit the sender has been notified of. Because of

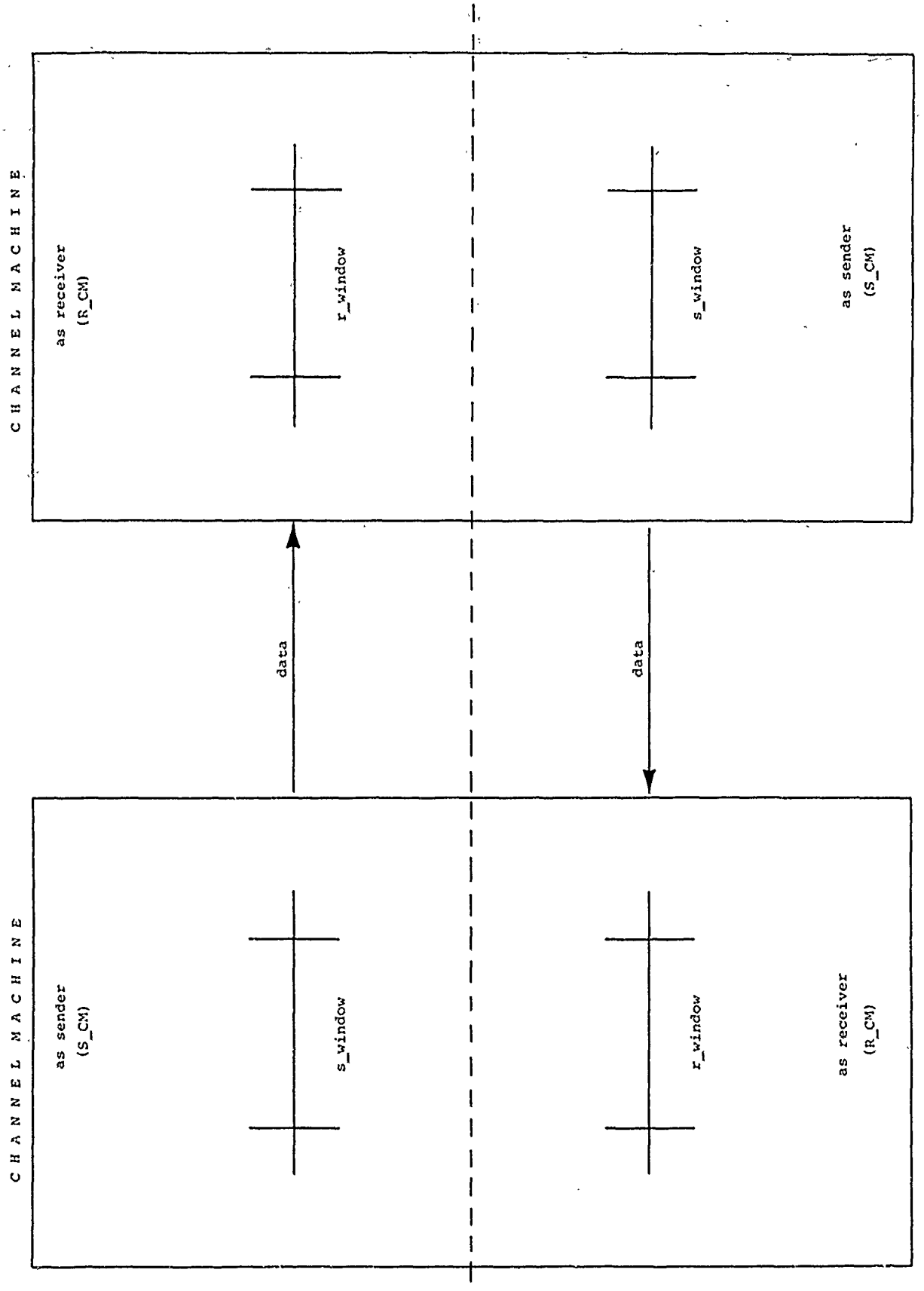


Figure 6: CHANNEL MACHINES WITH s_ and r_windows

messages in transit or messages lost, the sender's s_window may not agree with the receiver's r_window.

Since data can flow in both directions, each channel machine maintains both an r_window and an s_window (see Figure 6). For a receiving channel machine's r_window ("an R_CM's r_window"), the left edge represents the sequence number of the last acknowledgement sent to the sending channel machine ("the S_CM"). The right edge of the R_CM's r_window is computed by adding together modulo 16 the last amount of credit extended to the S_CM and the value of the left edge. This gives the largest sequence number that the R_CM is currently able to accept. Since the R_CM may receive several messages before it can acknowledge the first, the R_CM must keep track of the sequence number of the last message it has received. For an S_CM's s_window, the left edge represents the sequence number of the last acknowledgement received from the R_CM. The right edge is computed by adding together modulo 16 the Credit and Ack field of the last message received from the R_CM. This is the largest sequence number that the S_CM should send. Since the S_CM may send messages in advance of those that are acknowledged, the S_CM must keep track of the sequence number of the next message to be sent.

Duplicate Detection: duplicate Transmit_Commands may be introduced into the controlled data stream by retransmissions. If a Transmit_Command arrives at the R_CM with a sequence number less than the value of the left edge of its r_window,

the Transmit_Command is a duplicate. The duplicate is discarded and the S_CM is not notified.

Ordering: Transmit_Commands may arrive out-of-order due to events at the link level or due to retransmissions. If a Transmit_Command arrives with a sequence number more than one greater than the left edge of the R_CM's r_window, the Transmit_Command is out of order. The R_CM may keep the message if it has sufficient buffer space or it may discard it. In either case, the R_CM should send a Transmit_Response to the S_CM indicating that an out-of-order message was received (Status = 35) and with the Ack field set to the value of left edge of the R_CM's r_window. This will cause the S_CM to retransmit all Transmit_Commands from the sequence number of the Ack field in the Transmit_Response to the sequence number of the last message sent by the S_CM.

Flow Control: flow control must be provided both at the channel interface and between apposite channel machines. The channel interface events s_ready and ci_ready are used to control flow across the channel interface for a particular channel. The s_ready event indicates to the channel machine the number of ci_transmit_c events the SAPI is able to accept. The ci_ready event indicates to the SAPI the number of s_transmit_c events the channel machine is able to accept. It is assumed that events are not lost between the SAPI and the channel interface. The number of ci_transmit_c or s_transmit_c events allocated by each additional s_ready or

ci_ready event replaces the previous allocation.

Note: this particular channel interface model should not be viewed as an implementation specification. The channel interface model only specifies the properties that a channel interface must have. Many mechanisms may satisfy these properties.

When a channel machine receives a message from its apposite, it uses the Ack and Credit fields to update its s_window. The left edge of the s_window is set equal to the Ack field. The right edge is set equal to the sum modulo 16 of the Ack field plus the Credit field. The channel machine can now send messages to its apposite as long as the sequence numbers assigned are less than the value of the right edge of its s_window.

When a channel machine receives a Transmit_Command that is in order and not a duplicate, it advances the left edge of the r_window by one (modulo 16). It acknowledges this message (thereby acknowledging any others that have not been acknowledged as well). No more than 8 unacknowledged messages can be left outstanding. If the width of the r_window is near zero and the channel machine has sufficient allocation to pass data on to its SAPI, additional Credit should be extended to the sending channel machine.

Communicating Flow Control Information: although the controlled data stream consists solely of Transmit_Commands, other channel protocol messages carry flow control information. This section provides a table of the values of

the Seq, Ack, and Credit fields in these messages.

Begin_Command	Seq:	field used for Service Code
	Ack:	field used for Service Code
	Credit:	specifies the initial credit to the receiver
Begin_Response:	Seq:	zero
	Ack:	zero
	Credit:	specifies the initial credit to the receiver
Execute_Command:	Seq:	specifies the sequence number of the last Transmit_Command sent
	Ack:	specifies the sequence number of the last in order Transmit_Command correctly received
	Credit:	specifies the new credit value
Execute_Response:	Seq:	specifies the sequence number of the last Transmit_Command sent
	Ack:	specifies the sequence number of the last in order Transmit_Command correctly received
	Credit:	specifies the new Credit value
Transmit_Command:	Seq:	specifies the sequence number of this Transmit_Command
	Ack:	specifies the sequence number of the last in order Transmit_Command received correctly
	Credit:	specifies the new credit value
Transmit_Response:	Seq:	specifies the sequence number of the last Transmit_Command sent
	Ack:	specifies the sequence number of the last in order Transmit_Command received correctly
	Credit:	specifies the new Credit value
End_Command:	Seq:	specifies the sequence number

		of the last Transmit_Command sent
	Ack:	specifies the sequence number of the last Transmit_Command correctly received before the channel was terminated
	Credit:	irrelevant
End_Response:	Seq:	specifies the sequence number of the last Transmit_Command sent
	Ack:	specifies the sequence number of the last Transmit_Command correctly received before the channel was terminated
	Credit:	irrelevant

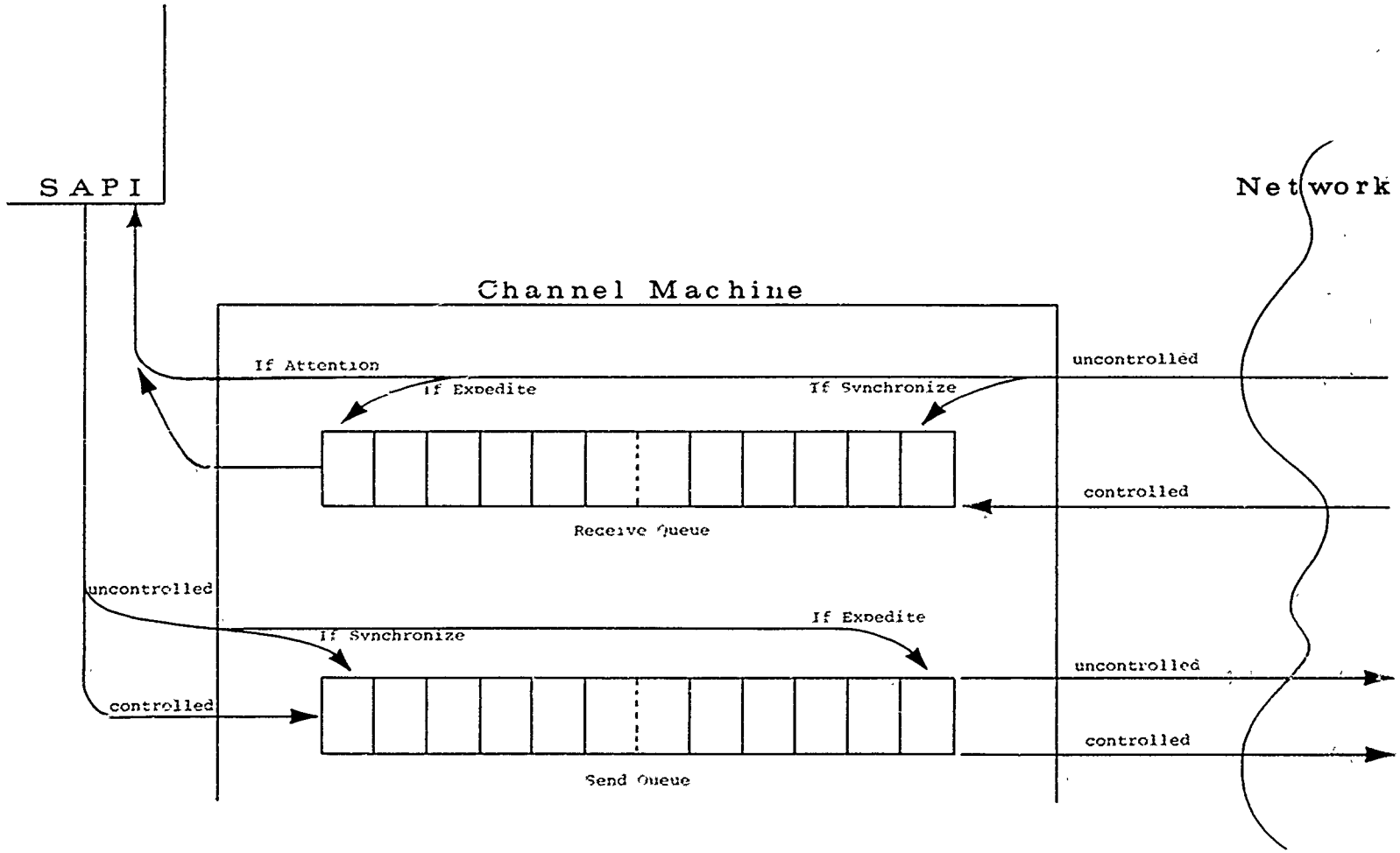


Figure 7: INTERACTION OF CONTROLLED AND UNCONTROLLED TRANSMISSION

Uncontrolled Transmission

The channel protocol implementation provides an uncontrolled data stream to apposite SAPIs. The uncontrolled data stream does not guarantee that order will be preserved, nor does it provide flow control. The uncontrolled data stream provides a means to expedite data transfer with respect to the controlled data stream. In addition, an "attention" to interrupt the SAPI can be associated with the expedited message. The uncontrolled data stream also provides a means to synchronize delivery of data in the uncontrolled data stream with the controlled data stream. Similarly, an attention to interrupt the SAPI can be associated with the synchronized message. The uncontrolled data stream is provided in the channel protocol by the Execute_Command and the Execute_Response. Figure 7 shows the interaction between controlled and uncontrolled transmission.

Expedited Data Flow: an Execute_Command may be sent with the Synchronize bit of the Control field set to zero. In this case, delivery of the Execute_Command is expedited. This means that the Execute_Command is placed at the head of the Sending Queue for delivery to the link level. When the Execute_Command arrives at the receiving channel machine its TEXT is placed at the head of the Receiving Queue for delivery to the SAPI.

Note: Whether or not Execute_Commands are expedited by the

link protocol implementation depends on the facilities provided by the link protocol. The channel protocol does not require that the link level provide this facility; however, it would be useful if available.

The Attention bit of the Control field of an expedited Execute_Command may be set to one. If the Attention bit is set, the receiving channel notifies the SAPI via an attention or an interrupt when the TEXT of the Execute_Command is placed at the head of its the Receiving Queue. The attention or interrupt is a special signal to the SAPI notifying it of important data waiting to be processed. If the Attention bit is set to zero, the channel machine places the TEXT of the Execute_Command at the end of its Receiving Queue and sends no attention to the SAPI. In this case the Execute_Command is synchronized (see below).

Synchronized Data Flow: an Execute_Command may be sent with the Synchronize bit of the Control field set to one. In this case, the sending channel machine places the Execute_Command at the end of its Sending Queue for delivery to the link level. The Execute_Command is sent to the apposite channel machine in the normal course of events and is not expedited. When the apposite channel machine receives the Execute_Command, it places the TEXT at the end of its Receiving Queue. If the Attention bit is set to one, it immediately notifies the SAPI by an interrupt or an attention that important data is waiting to be read. The

SAPI should process the queued data as quickly as possible in order to receive the TEXT of the Execute_Command and act on it.

Note: The effect of the Attention bit is not propagated ahead of a synchronized Execute_Command until it reaches the Receiving Queue. If such an effect is to be achieved, the sending SAPI should cause its channel machine to first send a synchronized Execute_Command with the Attention bit set, and then to send an expedited Execute_Command with the Attention bit set. This will cause an attention to be propagated ahead of the synchronized Execute_Command.

Termination

Channels may be terminated in two ways: flushing and non-flushing. A flushing termination causes all queued data to be discarded and causes the channel machine to enter a terminating state. A non-flushing termination allows all queued data to be sent before causing the channel machine to enter a terminating state.

Flushing Termination: when a channel machine is requested to perform a flushing termination, it discards all queued data in both the Receiving and Sending Queues and sends an End_Command. Any data tht arrives after the End_Command is sent is discarded. When the End_Command arrives at the apposite channel machine, all data in its Receiving Queue is discarded and the SAPI is notified by a channel interface event.

Non-flushing Termination: when a channel machine is requested to perform a non-flushing termination, the channel machine discards all data that was queued in its Receiving Queue for the SAPI, and enters an End_Command at the end of its Sending Queue. Any data that arrives after the End_Command has been entered in the Sending Queue is discarded. When the last Transmit_Command is sent, the channel machine then sends the End_Command and enters a terminating state. When the End_Command arrives at the apposite channel machine, it is placed at the end of the Receiving Queue. The End_Command is not acted upon until the last data has been delivered to the SAPI.

Non-Flushing End Deadlock Avoidance

Both of the apposite SAPIs using a virtual channel can simultaneously request non-flushing termination of the channel. Each channel machine must withhold sending the End_Command until the data which it has queued for sending to the other drains out. A deadlock will occur if neither channel machine can drain its Sending Queue of data for the other. This deadlock can be avoided by the following procedure.

If a channel machine is requested to perform a non-flushing termination, it shall:

1) discard all data queued in its Receiving Queue for the SAPI that requested the non-flushing termination.

2) continue to extend Credit to its apposite CPI (this allows its apposite to drain its Sending Queue of data toward it); and

3) if the channel machine does receive Transmit_Commands from its apposite, it shall acknowledge and discard them (since the SAPI has requested termination, the data need not be passed to it).

If the above procedure is followed, both of the apposite SAPIs may request non-flushing termination without a deadlock occurring.

BLANK PAGE

Notation and Nomenclature Conventions
for the
Channel Protocol

Notation

States: all state names are printed with all capital letters. If the state name consists of two or more words, the words are separated by an underscore (_). Examples: NULL, SENDER_PENDING.

Commands: all Command names are of the form:

<command name>_Command

where <command name> is one of the following:

Begin
End
Execute
Transmit

The first letter of each word is capitalized.
Examples: Begin_Command, End_Command.

Responses: all Response names are of the form:

<response name>_Response

where <response name> has the same range as <command name>. The first letter of each word is capitalized.
Examples: Begin_Response, End_Response.

Events: all names of events caused by the SAPI are either of the form:

s_<interface event name>_<event suffix>

where <interface event name> is one of the following:

begin
end
execute

transmit

and <event suffix> is either

	c	indicating a Command
or	r	indicating a Response

or of the form:

s_<interface event name>

where <interface event name> is one of the following:

identify
ready
status

All names of events caused by a channel machine are either of the form:

c_<interface event name>_<event suffix>

where <interface event name> is one of the following:

begin
end
execute
transmit

and <event suffix> is the same as above

or of the form:

c_<interface event name>

where <interface event name> is one of the following:

accept
ready
reject
status

Event names are all lower case. Examples: s_begin_c, c_begin_c, s_ready, c_status.

Nomenclature

Valid/Invalid: an invalid Command or Response is one that has failed to pass one or more consistency checks performed by the CPI. Most invalid Commands and Responses are detected by the multiplexor/demultiplexor and are handled at that time. Transmit_Commands may arrive out-of-order or may be outside the flow control window. These are also handled by the channel machine and are described in the state table.

Acceptable/Unacceptable: the channel machine checks events for consistency and accepts or rejects them via the c_accept or c_reject events.

Status ≠ 0: indicates that the Command corresponding to this Response was not successful. It indicates that the attempt to establish a virtual channel has failed.

Status = 2: indicates that the channel machine was not in a legal state to receive the Command corresponding to this Response.

Status = 39: indicates that the channel machine was in the SENDER_DRAINING state and discarded the Command corresponding to this Response.

Flushing/Non-Flushing: channels may be terminated in two ways:

> Flushing: any data queued for transmission is discarded at the time the termination is requested.
>
> Non-Flushing: the termination request is not acted on until all data queued for transmission has been sent (see Termination).

Log the error: if the channel machine detects an error, the error and any other diagnostic information should be written on a permanent file. In some cases, the error may be reported to the HFP Maintenance Service.

Channel Machine States

NULL: a channel machine in this state does not have an active channel.

SENDER_PENDING: a channel machine in this state is attempting to establish a channel. It has sent a Begin_Command and is waiting for a Begin_Response.

RECEIVER PENDING: a channel machine in this state has received a Begin_Command and is waiting for the s_begin_r event from the SAPI.

SENDER TAKING BACK: a channel machine in this state has been requested to terminate the channel while it was in the SENDER_PENDING state and has sent an End_Command before it has received the Begin_Response.

RECEIVER TAKING BACK: a channel machine in this state has received an End_Command while it was in the RECEIVER_PENDING state before the SAPI has responded to the c_begin_c event.

ESTABLISHED: a channel machine in this state has established a channel with its apposite.

SENDER DRAINING: a channel machine in this state has been requested to terminate the channel without flushing any data. The channel machine is sending all queued Transmit_Commands before sending the End_Response.

RECEIVER DRAINING: a channel machine in this state has received a non-flushing End_Command and is waiting until the SAPI has read all of the data queued for it before sending the End_Response.

SENDER TERMINATING: a channel machine in this state either

1) has been requested by the SAPI to terminate the channel and flush any data not yet sent, or

2) has been in the SENDER_DRAINING state, has sent the last Transmit_Command, and has also sent the End_Command.

RECEIVER TERMINATING: a channel machine in this state either

1) has received a flushing End_Command and is waiting for an s_end_r event from the SAPI, or

2) has been in the RECEIVER_DRAINING state, has passed the last c_transmit_c event and the c_end_c event to the SAPI, and is now waiting for the s_end_r event.

COMMANDS AND RESPONSES

Introduction

The following section defines the channel protocol commands and Responses. For each Command and each Response, there is a presentation of:

1. its function,
2. when it is sent,
3. the sending channel machine's state table for it,
4. the receiving channel machine's action upon receiving it
    a) in the normal case and
    b) in case of error,
5. the receiving channel machine's state table for it,
6. any subsequent action by the receiver
    a) in the normal case and
    b) in case of error,
7. any subsequent action by the sender
    a) in the normal case and
    b) in case of error,
8. the semantics of the fields of the HEADER for this Command or Response, and
9. the semantics of TEXT for this Command or Response.

The conventions followed in the state tables are given in the section entitled "Notation and Nomenclature Conventions for the Channel Protocol."

BLANK PAGE

Function

A Begin_Command is used by a channel machine to request its apposite to join it in establishing a host to front end channel.

When sent

When an acceptable s_begin_c event has been caused by an SAPI, the channel machine sends a Begin_Command.

Sending States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	acceptable s_begin_c	SENDER_ PENDING	c_accept Begin_Command	initialize channel machine

Action when received

In the normal case: a channel machine receiving a Begin_Command is in the NULL state. The channel machine then causes a c_begin_c event in order to notify the SAPI specified by the Service field of the Begin_Command that a channel to it has been requested and to pass to it the TEXT field of the Begin_Command. The channel machine then enters the RECEIVER_PENDING state.

In case of error: the channel machine logs the error (see HFP Maintenance Service), discards the message, and sends a Begin_Response with the Status code proper to the error (see Status codes for the Begin_Response, below).

Receiving States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	valid Begin_Command	RECEIVER_ PENDING	c_begin_c	initialize channel machine
any other state	Begin_Command (valid or invalid)	- - - - -	- - - - -	log the error

Subsequent action by the Receiver

In the normal case: an s_begin_r event with Status = Ø is caused by the SAPI in response to the c_begin_c. The channel machine then sends a Begin_Response with Status = Ø and enters the ESTABLISHED state.

In case of error: an s_begin_r with Status ≠ Ø event is caused by the SAPI. The channel machine then sends a Begin_Response with Status ≠ Ø and enters the NULL state.

Subsequent action by the Sender

In the normal case: the channel machine, having sent a Begin_Command, waits for a Begin_Response. If the channel machine receives a Begin_Response with Status = Ø, it then notifies the SAPI by causing a c_begin_r event with Status = Ø and enters the ESTABLISHED state.

In case of error: if the channel machine having sent a Begin_Command receives a Begin_Response with Status ≠ Ø, it notifies the SAPI of the error via a c_begin_r event with Status ≠ Ø and enters the NULL state.

Taking back: An SAPI having requested the channel machine to establish a channel may, via an s_end_c event, request the channel machine to terminate the channel before it receives the expected Begin_Response. In this case, the channel machine then sends an End_Command and enters the SENDER_TAKING_BACK state.

Semantics of fields

Type: Ø specifies Begin.

C/R: Ø specifies Command.

Credit: specifies the number of Transmit_Commands the sending channel machine is prepared to accept (see Transmission Control). Its value may be zero.

Service: specifies the SAPI to which the channel is to be established (see Message Multiplexing and Demultiplexing).

Group and Member: specifies the channel that is to be established.

Control: is currently undefined for the Begin_Command.

Semantics of TEXT

TEXT contains the service access protocol message.

Begin_Response

Function

A Begin_Response is used by a channel machine either to indicate the successful establishment of a host to front end channel as requested by its apposite or to indicate the reason for failure.

When Sent

When an acceptable s_begin_r event has been caused by an SAPI, the channel machine sends a Begin_Response.

Sending States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
RECEIVER_ PENDING	acceptable s_begin_r (Status=Ø)	ESTABLISHED	c_accept Begin_Response (Status=Ø)	
RECEIVER_ PENDING	acceptable s_begin_r (Status≠Ø)	NULL	c_accept Begin_Response (Status≠Ø)	
RECEIVER_ TAKING_BACK	acceptable s_begin_r (Status≠Ø)	NULL	c_accept Begin_Response (Status≠Ø)	

Action When Received

In the normal case: a channel machine receiving a Begin_Response with Status = Ø is in the SENDER_PENDING state. The channel machine then causes a c_begin_r event with Status = Ø in order to notify the SAPI that the channel has been established and to pass to it the TEXT field of the Begin_Response. The channel machine then enters the ESTABLISHED state.

In case of error: if a channel machine receiving a Begin_Response with Status ≠ Ø is in either the SENDER_PENDING state or the SENDER_TAKING_BACK state, it causes a c_begin_r event with Status ≠ Ø in order to pass the TEXT field of the Begin_Response to the SAPI and enters the NULL state; if a channel machine receiving a Begin_Response with Status = Ø is

in the SENDER_TAKING_BACK state, it takes no action and does not change state; if a channel machine receiving a Begin_Response is in neither the SENDER_PENDING state nor the SENDER_TAKING_BACK state, it logs the error (see HFP Maintenance Service) but does not change state; if an inconsistency in the Begin_Response has been detected, the filter logs the error (see Message Formatting and Filtering, HFP Maintenance Service).

Receiving States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
SENDER_ PENDING	valid Begin_Response (Status=0)	ESTABLISHED	c_begin_r (Status=0)	
SENDER_ PENDING	valid Begin_Response (Status≠0)	NULL	c_begin_r (Status≠0)	- - - - -
SENDER_ PENDING	invalid Begin_Response	SENDER_ TERMINATING	c_begin_r (Status≠0) End_Command (flushing)	log the error
SENDER_ TAKING_BACK	valid Begin_Response (Status=0)	- - - - -	- - - - -	
SENDER_ TAKING_BACK	valid Begin_Response (Status≠0)	NULL	c_begin_r (Status≠0)	
any other state	Begin_Response (valid or invalid)	- - - - -	- - - - -	log the error

Subsequent Action by the Receiver

In the normal case: the channel machine exchanges data with its apposite.

In case of error: none if the channel machine had been in either the SENDER_PENDING state or the SENDER_TAKING_BACK state; had it been in any other state, it logs the error but does change state.

Subsequent Action by the Sender

In the normal case: the channel machine exchanges data with its apposite.

In case of error: The channel machine returns to the NULL state.

Semantics of Fields

Type: Ø specifies Begin.

C/R: 1 specifies Response.

Credit: specifies the number of Transmit_Commands the sender of the Begin_Response is prepared to accept (see Transmission Control). If there was an error, the content of this field is irrelevant.

Seq: is zero. If there was an error, the content of this field is irrelevant.

Ack: specifies zero as the sequence number of the last message correctly received. If there was an error, the content of this field is irrelevant.

Group and Member: specifies the channel which the Begin_Command requested to be established.

Status indicates the success or failure of the attempt to establish the channel. The following Status codes are applicable to the Begin_Response:

Status	Meaning
Ø	Command was successful.
1	Channel non-existent: the Group and Member fields of the Begin_Command referenced a channel machine unknown to the receiving CPI.
2	Illegal state: the Begin_Command referenced a channel machine which was in a state for which the Begin_Command is an illegal input.
3	Command not implemented: the Begin_Command is not implemented by the receiving CPI.
5	Message too long: the number of bits in the Begin_Command exceeded the maximum permitted by the receiving CPI.

6 Service access protocol message error: an error in the service access protocol message contained in the TEXT field of the Begin_Command was detected by the SAPI.

32 Channel in use: the channel referenced in the Begin_Command was already assigned (i.e., not in the NULL state).

33 Service not implemented: the Service field in the Begin_Command specified an SAPI not implemented at the receiving site.

34 Insufficient resources: the receiver of the Begin_Command did not have sufficient resources for establishing the host to front end channel.

37 Bad channel polarity: the high-order bit of the Group field in the Begin_Command had the wrong value.

38 Service not operational: the Service field in the Begin_Command specified an SAPI which is implemented at the receiving site but which is temporarily unavailable.

Semantics of TEXT

TEXT contains the service access protocol message.

End_Command

Function

An End_Command is used by a CPI to request its apposite CPI to join it in terminating a host to front end channel, a group of channels, or all channels.

A CPI sending an End_Command has two options:

1. it may request that the channel(s) be terminated immediately ("flushing termination");

2. it may request that the channel(s) be terminated only after any data queued by its apposite CPI has been passed on to the apposite CPI's SAPI(s) ("non-flushing termination").

(For further discussion also see Termination.)

When Sent

When an acceptable s_end_c event has been caused by an SAPI, its CPI sends an End_Command.

Sending States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	acceptable s_end_c (flushing, from HFP Maintenance Service)	- - - - -	c_accept End_Command (flushing)	
SENDER_ PENDING	acceptable s_end_c (flushing or non-flushing)	SENDER_ TAKING_BACK	c_accept End_Command (flushing)	
SENDER_ PENDING	invalid Begin_Response	SENDER_ TERMINATING	c_begin_r (Status≠0) End_Command (flushing)	log the error
RECEIVER_ PENDING	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	
SENDER_ TAKING_BACK	acceptable s_end_c (flushing or non-flushing)	- - - - -	c_accept End_Command (flushing)	
RECEIVER_ TAKING_BACK	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	
ESTABLISHED	acceptable s_end_c (flushing)	SENDER_ TERMINATING	c_accept End_Command (flushing)	discard any queued data
ESTABLISHED (with data queued for apposite CPI)	acceptable s_end_c (non-flushing)	SENDER_ DRAINING	c_accept	discard any data queued for SAPI; send data to apposite CPI; place End_Command (non-flushing) at end of send queue; acknowledge but
ESTABLISHED (with no data queued for apposite CPI)	acceptable s_end_c (non-flushing)	SENDER_ TERMINATING	c_accept End_Command (non-flushing)	discard any data queued for SAPI; acknowledge but discard any incoming messages
SENDER_ DRAINING	acceptable s_end_c (flushing)	SENDER_ TERMINATING	c_accept End_Command (flushing)	discard any queued data
SENDER_ DRAINING	(acknowledgement of last Transmit_ Command	SENDER_ TERMINATING	End_Command (non-flushing)	
RECEIVER_ DRAINING	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	discard any queued data
SENDER_ TERMINATING	acceptable s_end_c (flushing or non-flushing)	- - - - -	c_accept End_Command (flushing)	
RECEIVER_ TERMINATING	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	

Action When Received

In the normal case: a channel machine receiving an End_Command is in either the SENDER_PENDING state, the ESTABLISHED state, the SENDER_DRAINING state, or the SENDER_TERMINATING state. If the End_Command specifies a non-flushing termination, the channel machine first waits until any data queued for the SAPI has been read by the SAPI (see Non-Flushing End Deadlock Avoidance). If the End_Command specifies a flushing termination, the channel machine discards any data queued for the SAPI. The channel machine then causes a c_end_c event in order to pass the TEXT field of the End_Command to the SAPI. The channel machine then enters the RECEIVER_TAKING_BACK state if it had been in the RECEIVER_PENDING state, the RECEIVER_TERMINATING state if it had been in the RECEIVER_DRAINING state, or the NULL state if it had been in the SENDER_TERMINATING state.

In case of error: the channel machine logs the error (see HFP Maintenance Service), discards the message, and sends an End_Response with the Status code proper to the error (see Status codes for End_Response, below).

Receiving States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	End_Command (valid or invalid, flushing or non-flushing)	- - - - -	End_Response	discard the message
SENDER_ PENDING	End_Command (valid or invalid, flushing or non-flushing)	NULL	c_end_c End_Response	
RECEIVER_ PENDING	valid End_Command	RECEIVER_ TAKING_BACK	c_end_c	
RECEIVER_ PENDING	invalid End_Command	NULL	c_end_c End_Response	log the error
SENDER_ TAKING_BACK	valid End_Command	NULL	c_end_c End_Response	
RECEIVER_ TAKING_BACK	valid End_Command	NULL	c_end_c End_Response	
ESTABLISHED	valid End_Command (flushing)	RECEIVER_ TERMINATING	c_end_c	discard any data queued for SAPI and for apposite CPI
ESTABLISHED (with data queued for SAPI)	valid End_Command (non-flushing)	RECEIVER_ DRAINING	- - - - -	discard any data queued for apposite CPI; place c_end_c at end of receive queue; acknowledge but discard any incoming messages
ESTABLISHED (with no data queued for SAPI)	valid End_Command (non-flushing)	RECEIVER_ TERMINATING	c_end_c	discard any data queued for apposite CPI; acknowledge but discard any incoming messages
ESTABLISHED	invalid End_Command (flushing or non-flushing)	NULL	c_end_c End_Response	log the error
SENDER_ DRAINING	valid End_Command (flushing)	NULL	c_end_c End_Response	discard any data queued
RECEIVER_ DRAINING	valid End_Command (flushing)	RECEIVER_ TERMINATING	c_end_c	discard any data queued
SENDER_ TERMINATING	valid End_Command (flushing or non-flushing)	NULL	c_end_c	
SENDER_ TERMINATING	invalid End_Command	NULL	c_end_c End_Response	log the error
RECEIVER_ TERMINATING	valid End_Command (flushing or non-flushing)	NULL	c_end_c End_Response	

Subsequent Action by the Receiver

In the normal case: a s_end_r event with Status = Ø is caused by the SAPI in response to the c_end_c event. The channel machine then sends an End_Response with Status = Ø and enters the NULL state.

In case of error: a s_end_r event with Status ≠ Ø is caused by the SAPI. The channel machine then sends an End_Response with Status ≠ Ø and enters the NULL state.

Subsequent Action by the Sender

In the normal case: if a non-flushing termination was specified, the channel machine continues to acknowledge Transmit_Commands, to extend flow-control Credit, to act upon any flow-control information contained in in-coming Commands or Responses, and to discard any incoming Commands or Responses until it receives an End_Command or an End_Response.

In case of error: the channel machine notifies the SAPI of the error via a c_end_r event with Status ≠ Ø which passes the TEXT field of the End_Response to the SAPI. The channel machine then enters the NULL state.

Semantics of Fields

Type: 4 specifies End.

C/R: Ø specifies Command.

Credit: is irrelevant.

Seq: specifies the sequence number of the last Transmit_Command sent by the sender of the End_Command.

Ack: specifies the sequence number of the last Transmit_Command correctly received before the channel was terminated.

Group and Member: specifies the channel(s) to be terminated. If Group is not zero and Member is zero, all channels with the same Group are to be terminated. If both Group and Member are zero, all channels are to be terminated. The latter option is intended as part of the restart sequence (see Initializing Host - Front End Communication).

Control The Control field bits have the following meanings:

Flush away: (bit 1 = 1) means immediately flush data which is queued going away from the sender of the End_Command and terminate the channel. If this option is not requested (i.e., if bit 1 = Ø), the End_Command is not to take effect until all data queued going away from the sender of the End_Command has been processed in the normal manner.

Semantics of TEXT

TEXT contains the service access protocol message.

End_Response

Function

An End_Response is used by a channel machine to acknowledge to its apposite that a channel, group of channels, or all channels have been terminated as requested by its apposite.

When Sent

When an acceptable s_end_r event has been caused by an SAPI, its CPI sends an End_Response.

Sending States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	acceptable s_end_r (from HFP Maintenance Service)	- - - - -	c_accept End_Response	
NULL	End_Command (valid or invalid, flushing or non-flushing)	- - - - -	End_Response	discard the message
SENDER_ PENDING	End_Command (valid or invalid, flushing or non-flushing)	NULL	c_end_c End_Response	
RECEIVER_ PENDING	invalid End_Command	NULL	c_end_c End_Response	log the error
SENDER_ TAKING_BACK	valid End_Command	NULL	c_end_c End_Response	
RECEIVER_ TAKING_BACK	acceptable s_end_r	NULL	c_accept End_Response	
RECEIVER_ TAKING_BACK	valid End_Command	NULL	c_end_c End_Response	
ESTABLISHED	invalid End_Command (flushing or non-flushing)	NULL	c_end_c End_Response	log the error
SENDER_ DRAINING	valid End_Command (flushing)	NULL	c_end_c End_Response	discard any data queued
SENDER_ TERMINATING	invalid End_Command	NULL	c_end_c End_Response	log the error
RECEIVER_ TERMINATING	acceptable s_end_r	NULL	c_accept End_Response	
RECEIVER_ TERMINATING	valid End_Command (flushing or non-flushing)	NULL	c_end_c End_Response	

Action When Received

In the normal case: a channel machine receiving an End_Response is in either the SENDER_TAKING_BACK state or the SENDER_TERMINATING state. The channel machine then causes a c_end_r event in order to pass to the TEXT field of the End_Response to the SAPI. The channel machine then enters the NULL state.

In case of error: the channel machine is to log the error (see HFP Maintenance Service) and, if appropriate, cause a c_end_r event in order to pass the TEXT field of the End_Response to the SAPI. In either case, the channel machine then enters the NULL state.

Receiving States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	End_Response	- - - - -	- - - - -	discard the message
SENDER_ TAKING_BACK	valid End_Response	NULL	c_end_r	
SENDER_ TERMINATING	valid End_Response	NULL	c_end_r	
any other state	End_Response (valid or invalid)	- - - - -	- - - - -	log the error

Subsequent Action by the Receiver

In the normal case: none.

In case of error: none.

Subsequent Action by the Sender

In the normal case: none.

In case of error: the channel machine logs the error and enters the NULL state.

Semantics of Fields

Type: 4 specifies End.

C/R: 1 specifies Response.

Credit: is irrelevant.

Seq: specifies the sequence number of the last Transmit_Command sent by the sender of the End_Response.

Ack: specifies the sequence number of the last Transmit_Command sent by the sender of the End_Response.

Group and Member: specifies the channel(s) referenced in the End_Command.

Status: indicates whether or not an error has been encountered. The following codes are applicable to the End_Response:

Status	Meaning
Ø	Command was successful.
1	Channel non-existent: the Group and Member field of the End_Command referenced a channel machine unknown to the receiving CPI.
4	Option not implemented: a Control field option was specified in the End_Command which is legal but not implemented by the receiving CPI.
5	Message too long: the number of bits in the End_Command exceeded the maximum permitted by the receiving CPI.
6	Service access protocol message error: an error in the service access protocol message contained in the TEXT field of the End_Command was detected by the SAPI.
7	Illegal Control field value: the Control field of the End_Command contained an undefined value.

End_Response

Semantics of TEXT

TEXT contains the service access protocol message.

Execute_Command

Function

An Execute_Command is used by a channel machine to effect the transfer of data over a channel to its apposite without the guarantees of flow-control and order provided via Transmit_Command (see Overview, Transmission Control).

An SAPI requesting a channel machine to send an Execute_Command has three options:

1. it may request that the Execute_Command be delivered asynchronously vis-a-vis the flow-controlled, in-order data stream,

2. it may request that the Execute_Command be delivered in synchronization with a specified point in the flow-controlled, in-order data stream;

3. in either case, it may request that the Execute_Command carry with it a request that its apposite SAPI be notified of its arrival.

When Sent

When an acceptable s_execute_c event has been caused by an SAPI, the channel machine sends an Execute_Command.

Sending States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	acceptable s_execute_c (synchronize)	- - - - -	c_accept Execute_Command	place Execute_Command at end of send queue
ESTABLISHED	acceptable s_execute_c (expedite)	- - - - -	c_accept Execute_Command	place Execute_Command at head of send queue

Action When Received

In the normal case: the channel machine receiving an Execute_Command is in the ESTABLISHED state. If the Synchronize bit is not set, the channel machine then causes, at the earliest opportunity, a c_execute_c event in order to pass the TEXT field of the Execute_Command directly to the SAPI, i.e., ahead of any other data queued for it. If the Synchronize bit is set, the channel machine will deliver the Execute_Command at the point in the data stream where it was received. If the Attention bit is set, the SAPI is notified immediately via the a system specific attention or out-of-band signal.

In case of error: the channel machine logs the error (see HFP Maintenance Service), discards the message, and sends an Execute_Response with the Status code proper to the error (see Status codes for Execute_Response below).

Receiving States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	valid Execute_Command (synchronize)	- - - - -	c_execute_c	place c_execute_c at end of receive queue
ESTABLISHED	valid Execute_Command (synchronize, attention)	- - - - -	c_execute_c	notify SAPI of attention, place c_execute_c at end of receive queue
ESTABLISHED	valid Execute_Command (expedite)	- - - - -	c_execute_c	place c_execute_c at head of receive queue
ESTABLISHED	valid Execuce_Command (expedite, attention)	- - - - -	c_execute_c	notify SAPI of attention, place c_execute_c at head of receive queue
SENDER_ DRAINING	valid Execute_Command	- - - - -	Execute_Response (Status=39)	acknowledge but discard the message
any other state	Execute_Command (valid or invalid)	- - - - -	- - - - -	log the error

Subsequent Action By the Receiver

In the normal case: an s_execute_r event with Status = Ø is caused by the SAPI in response to the c_execute_c event. The channel machine then sends an Execute_Response with Status = Ø, and continues to exchange data with its apposite.

In case of error: a s_execute_r event with Status ≠ Ø is caused by the SAPI. The channel machine then sends an Execute_Response with Status ≠ Ø and resumes data exchange with its apposite.

Subsequent Action by the Sender

In the normal case: the channel machine continues to exchange data with its apposite.

In case of error: if the channel machine receives an Execute_Response with Status ≠ Ø, it notifies the SAPI of the error via a c_execute_r event with Status ≠ Ø and resumes data exchange with its apposite.

Semantics of Fields

Type: 3 specifies Execute.

C/R: Ø specifies Command.

Credit: specifies the number of Transmit_Commands beyond the number specified by the Ack field, which the sender of the Transmit_Response is prepared to receive.

Seq: specifies the sequence number of the last Transmit_Command sent by the sender of the Execute_Command.

Ack: specifies the sequence number of the last in-sequence Transmit_Command correctly received by the sender of the present Execute_Command.

Group and Member: specifies the channel over which the Execute_Command is to be sent.

Control: the Control field bits have the following meaning:

Bit Ø123	Meaning
ØØØØ	Place the Execute_Command at the head of the data queue.

1000	Place the Execute_Command at the head of the data queue. Notify the SAPI of the Attention.
0001	Place the Execute_Command at the end of the data queue.
1001	Place the Execute_Command at the end of the data queue. Notify the SAPI of the Attention.

Semantics of TEXT

TEXT contains the service access protocol message.

Execute_Response

Function

An Execute_Response is used by a channel machine to effect the transfer of data to its apposite in response to an Execute_Command. Like the Execute_Command, the Execute_Response is not subject to flow-control and order guarantee.

When Sent

When an acceptable s_execute_r event has been caused by an SAPI, the channel machine sends an Execute_Response.

Sending States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	acceptable s_execute_r	- - - - -	c_accept Execute_Response	place Execute_Response at head of send queue
SENDER_ DRAINING	valid Execute_Command	- - - - -	Execute_Response (Status=39)	acknowledge but discard the message

Action When Received

In the normal case: a channel machine receiving an Execute_Response is in the ESTABLISHED state. The channel machine then causes, at the earliest opportunity, a c_execute_r event in order to pass the TEXT field of the Execute_Response directly to the SAPI, i.e., ahead of any other data queued for it. The channel machine does not change state.

In case of error: the channel machine logs the error (see HFP Maintenance Service), notifies the SAPI of the error via a c_execute_r event which also passes the TEXT field of the Execute_Response to the SAPI. The channel machine does not change state.

Receiving States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	valid Execute_Response	- - - - -	c_execute_r	place c_execute_r at head of receive queue
SENDER_ DRAINING	valid Execute_Response	- - - - -	- - - - -	discard the message
any other state	Execute_Response (valid or invalid)	- - - - -	- - - - -	log the error

Subsequent Action by the Receiver

In the normal case: the channel machine continues to exchange data with its apposite.

In case of error: the channel machine resumes data exchange with its apposite.

Subsequent Action by the Sender

In the normal case: the channel machine continues to exchange data with its apposite.

In case of error: the channel machine resumes data exchange with its apposite.

Semantics of Fields

Type: 3 specifies Execute.

C/R: 1 specifies Response.

Credit: specifies the number of Transmit_Commands beyond the number specified by the Ack field, which the sender of the Transmit_Response is prepared to receive.

Seq: specifies the sequence number of the last Transmit_Command sent by the sender of the Execute_Response.

Ack: specifies the sequence number of the last in-sequence Transmit_Command correctly received by the sender of the present Execute_Response.

Group and Member: specifies the channel over which the Execute_Response is to be sent.

Status: indicates whether or not an error has been encountered. The following codes are applicable to the Execute_Response:

Status	Meaning
Ø	Command was successful.
1	Channel non-existent: the Group and Member fields of the Execute_Command referenced a channel machine unknown to the receiving CPI.
2	Illegal state: the Execute_Command referenced a channel machine which was in a state for which the Execute_Command is an illegal input.
3	Command not implemented: the Execute_Command is not implemented by the receiving CPI.
5	Message too long: the number of bits in the Execute_Command exceeded the maximum permitted by the receiving CPI.
6	Service access protocol message error: an error in the service access protocol message contained in the TEXT field of the Execute_Command was detected by the SAPI.
7	Illegal Control field value: the Control field of the Execute_Command contained an undefined value.
39	Command discarded: the channel machine is in the SENDER_DRAINING or SENDER_TERMINATING state, and has discarded the Execute_Command without passing it to the service access level.

Semantics of TEXT

TEXT contains the service access protocol message.

NOP

Function

A NOP is used by a channel machine as a filler when a channel protocol message does not completely fill a link level protocol frame.

When Sent

When a channel protocol message does not completely fill a link protocol frame, the channel machine may send a NOP as filler.

Sending States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
any state	any input	- - - - -	NOP	

Action When Received

In the normal case: the channel discards the NOP.

Receiving States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
any state	NOP	- - - - -	- - - - -	discard

Subsequent Action by the Receiver

None

Subsequent Action by the Sender

None

Semantics of Fields

Type: 5 specifies NOP.

All other fields are irrelevant. These fields may contain any values.

Semantics of TEXT

None.

Transmit_Command

Function

A Transmit_Command is used by a channel machine to effect the flow-controlled, in-order transfer of data to its apposite (see Overview, Transmission Control).

When Sent

When an acceptable s_transmit_c event has been caused by an SAPI, the channel machine sends a Transmit_Command (flow-control permitting).

Sending States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	acceptable s_transmit_c	- - - - -	c_accept Transmit_Command	

Action When Received

In the normal case: a channel machine receiving a Transmit_Command is in either the ESTABLISHED state or the RECEIVER_DRAINING state. The channel machine then causes a c_transmit_c event in order to pass the TEXT field of the Transmit_Command to the SAPI. The channel machine does not change state, but does apply the transmission control discipline (see Transmission Control). Transmit_Commands may also be received in the RECEIVER_TERMINATING state after a non-flushing End_Command has been received. (see Termination, End_Command).

In case of error: the channel machine logs the error (see HFP Maintenance Service), discards the message, and sends a Transmit_Response with the Status code proper to the error (see Status for Transmit_Response below).

Receiving States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED (with data queued for apposite CPI)	valid Transmit_Command	- - - - -	c_transmit_c Transmit_Command(s)	update flow control
ESTABLISHED (with no data queued for apposite CPI)	valid Transmit_Command	- - - - -	c_transmit_c Transmit_Response	update flow control
ESTABLISHED	invalid Transmit_Command	- - - - -	Transmit_Response (Status≠Ø)	log the error
SENDER_ DRAINING	valid Transmit_Command	- - - - -	Transmit_Response (Status=39)	acknowledge but discard any incoming messages
SENDER_ DRAINING	invalid Transmit_Command	- - - - -	Transmit_Response (Status≠Ø)	log the error
SENDER_ TERMINATING	valid Transmit_Command	- - - - -	- - - - -	discard the message
any other state	Transmit_Command (valid or invalid)	- - - - -	- - - - -	log the error

Subsequent Action by the Receiver

In the normal case: the channel machine acknowledges its receipt of the Transmit_Command via the Ack field of the next Command or Response it sends, and does not change state. When flow control credit must be extended and the channel machine has no other messages to send, it sends a Transmit_Response.

In case of error: the channel machine sends a Transmit_Response with Status ≠ Ø but does not change state.

Subsequent Action by the Sender

In the normal case: as long as the channel machine has data to send and flow control permits, the channel machine continues to exchange data with its apposite.

In case of error: the channel machine logs the error and takes the appropriate action, resuming data exchange with its apposite.

Semantics of Fields

Type: 1 specifies Transmit.

C/R: Ø specifies Command.

Credit: specifies the number of Transmit_Commands beyond the number specified by the ACK field, which the sender of the Transmit_Command is prepared to receive.

Seq: specifies the sequence number of this Transmit_Command.

Ack: specifies the sequence number of the last in-sequence Transmit_Command correctly received by the sender of the present Transmit_Command.

Group and Member: specifies the channel over which the Transmit_Command is to be sent.

Control The use of this field is undefined.

Semantics of TEXT

TEXT contains the service access protocol message.

Transmit_Response

Function

A Transmit_Response is used by a channel machine to pass transmission control information to its apposite (e.g., to acknowledge receipt of a Transmit_Command, to update flow-control) or to report to its apposite an error in a Transmit_Command received.

When Sent

When a channel machine must acknowledge a Transmit_Command and has no Transmit_Commands to send, or when an error has been detected in the Transmit_Command, or when flow-control Credit (see Transmission Control) must be extended and the channel machine has no other messages to send, it sends a Transmit_Response.

Sending States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED (with no data queued for apposite CPI)	valid Transmit_Command	- - - - -	c_transmit_c Transmit_Response	update flow control
ESTABLISHED	invalid Transmit_Command	- - - - -	Transmit_Response (Status≠0)	log the error
SENDER_DRAINING	valid Transmit_Command	- - - - -	Transmit_Response (Status=39)	acknowledge but discard any incoming messages
SENDER_DRAINING	invalid Transmit_Command	- - - - -	Transmit_Response (Status≠0)	log the error

Action When Received

In the normal case: a channel machine receiving a Transmit_Response is in either the ESTABLISHED state, the SENDER_DRAINING state, or the SENDER_TERMINATING state. The channel machine does not change state, but does apply the transmission control discipline (see Transmission Control).

In case of error: the channel machine logs the error (see HFP Maintenance Service), and performs any necessary error-recovery, but does not change state. If the error was "message out of order" (Status = 35), the channel machine is to retransmit all messages up to and including the last message sent.

Receiving States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	valid Transmit_Response	- - - - -	- - - - -	update flow control; log any error
SENDER_ DRAINING	valid Transmit_Response	- - - - -	- - - - -	update flow control
SENDER_ TERMINATING	valid Transmit_Response	- - - - -	- - - - -	update flow control
any other state	Transmit_Response (valid or invalid)	- - - - -	- - - - -	log the error

Subsequent Action By the Receiver

In the normal case: the channel machine continues to exchange data with its apposite.

In case of error: the channel machine resumes data exchange with its apposite.

Subsequent Action by the Sender

In the normal case: the channel machine continues to exchange data with its apposite.

In case of error: the channel machine resumes data exchange with its apposite. (If the channel machine encounters a high frequency of erroneous Transmit_Commands some special action may be required.)

Semantics of Fields

Type: 1 specifies Transmit.

C/R: 1 specifies Response.

Credit: specifies the number of Transmit_Commands beyond the number specified by the Ack field, which the sender of the Transmit_Response is prepared to receive.

Seq: specifies the sequence number of the last Transmit_Command sent by the sender of the Transmit_Response.

Ack: specified the sequence number of the last in-sequence Transmit_Command correctly received by the sender of the present Transmit_Response.

Group and Member: specifies the channel over which Transmit_Response is to be sent.

Status: indicates whether or not an error has been encountered. The following codes are applicable to the Transmit_Response:

Status	Meaning
Ø	Command was successful.
1	Channel non-existent: the Group and Member fields of the Transmit_Command referenced a channel machine unknown to the receiving CPI.
2	Illegal state: the Transmit_Command has referenced a channel machine which was in a state for which the Transmit_Command is an illegal input.
5	Message too long: the number of bits in the Transmit_Command exceeded the maximum permitted by the receiving CPI.

6 Service access protocol message error: an error in the service access protocol message contained in the TEXT field of the Transmit_Command was detected by the SAPI.

35 Out of sequence: a Transmit_Command was received and discarded whose Seq field was neither in sequence [equal to <last received> + 1] nor a duplicate [between (<last received> - 7) and <last received> inclusive] (see Transmission Control).

36 Out of window: a Transmit_Command was received and discarded whose Seq field was between (<last received> + Credit + 1) and (<last received> + 8) inclusive (see Transmission Control).

39 Command discarded: the channel machine received a Transmit_Command or an Execute_Command, is in the SENDER_DRAINING or SENDER_TERMINATING state, and has discarded the Command without passing it to the service access level.

Semantics of TEXT

undefined

Field	Function
Size	specifies the number of bits in the entire message.
Type	specifies the message type: Begin 0 Transmit 1 Execute 3 End 4 Nop 5
C/R	(C/R = 0) indicates a Command or a (C/R = 1) indicates a Response.
Credit	specifies the number of Transmit_Commands beyond the number specified by the Ack field, which the sender of this message is prepared to receive.
Seq	specifies, in a Transmit_Command its sequence number.
Ack	specifies the sequence number of the last in-sequence Transmit_Command correctly received by the sender of this message.
Service	specifies, in Begin_Command, the SAPI to which the channel is to be established.
Group	specifies the channel group which the message references.
Member	specifies the channel which the message references within the channel group.

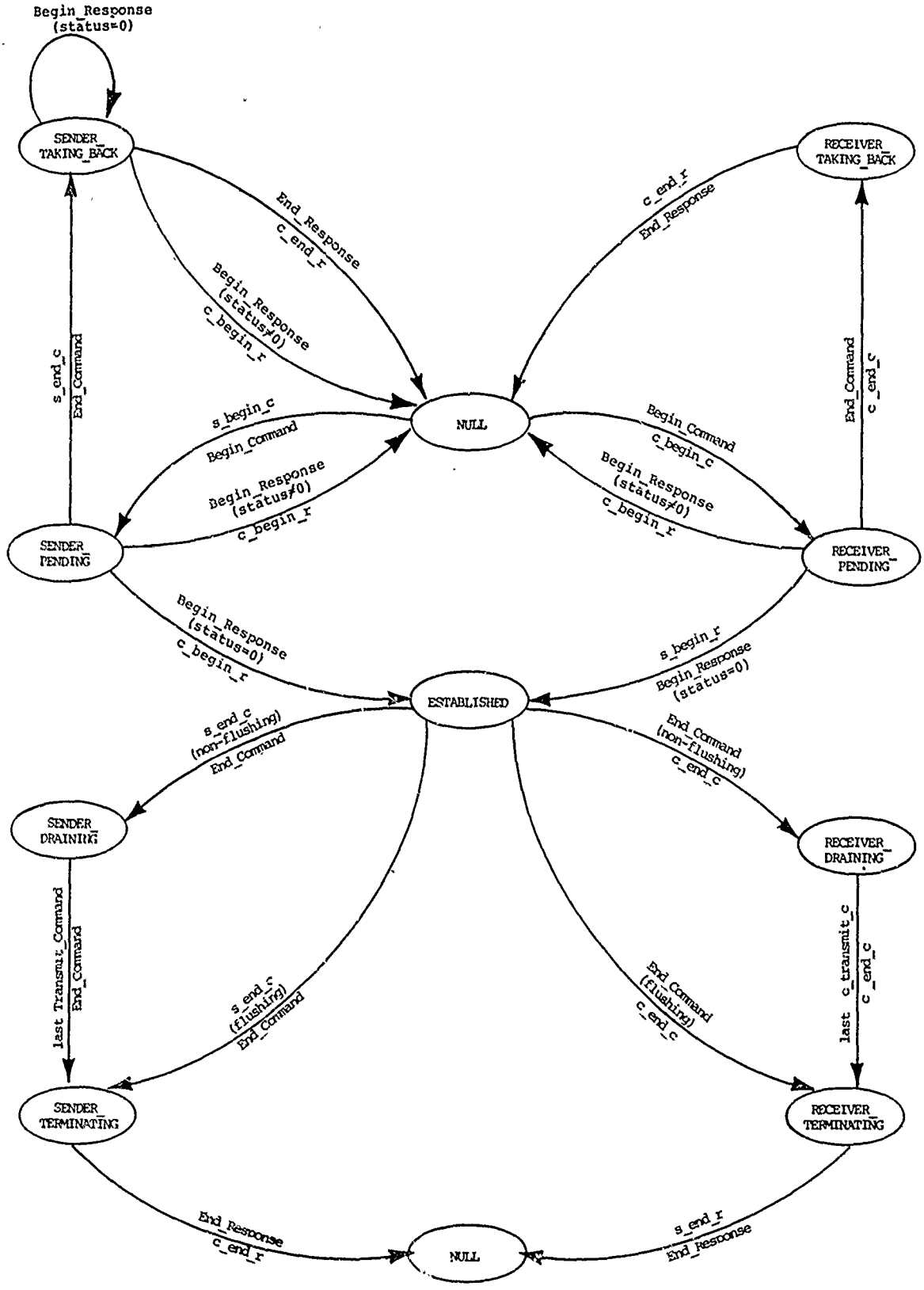
Message Format

Field Name	Field Size (bits)	Alternate Field Size (bits)	Alternate Field Name
HEADER			
Size	16		
Type	3		
C/R	1		
Credit	4		
Seq	4		
Ack	4	8	Service (BEGIN Command)
(not used)	4		
Group	12		
Member	16		
Control (Commands)	8	8	Status (Responses)
PAD	Note A		
TEXT	Note B		

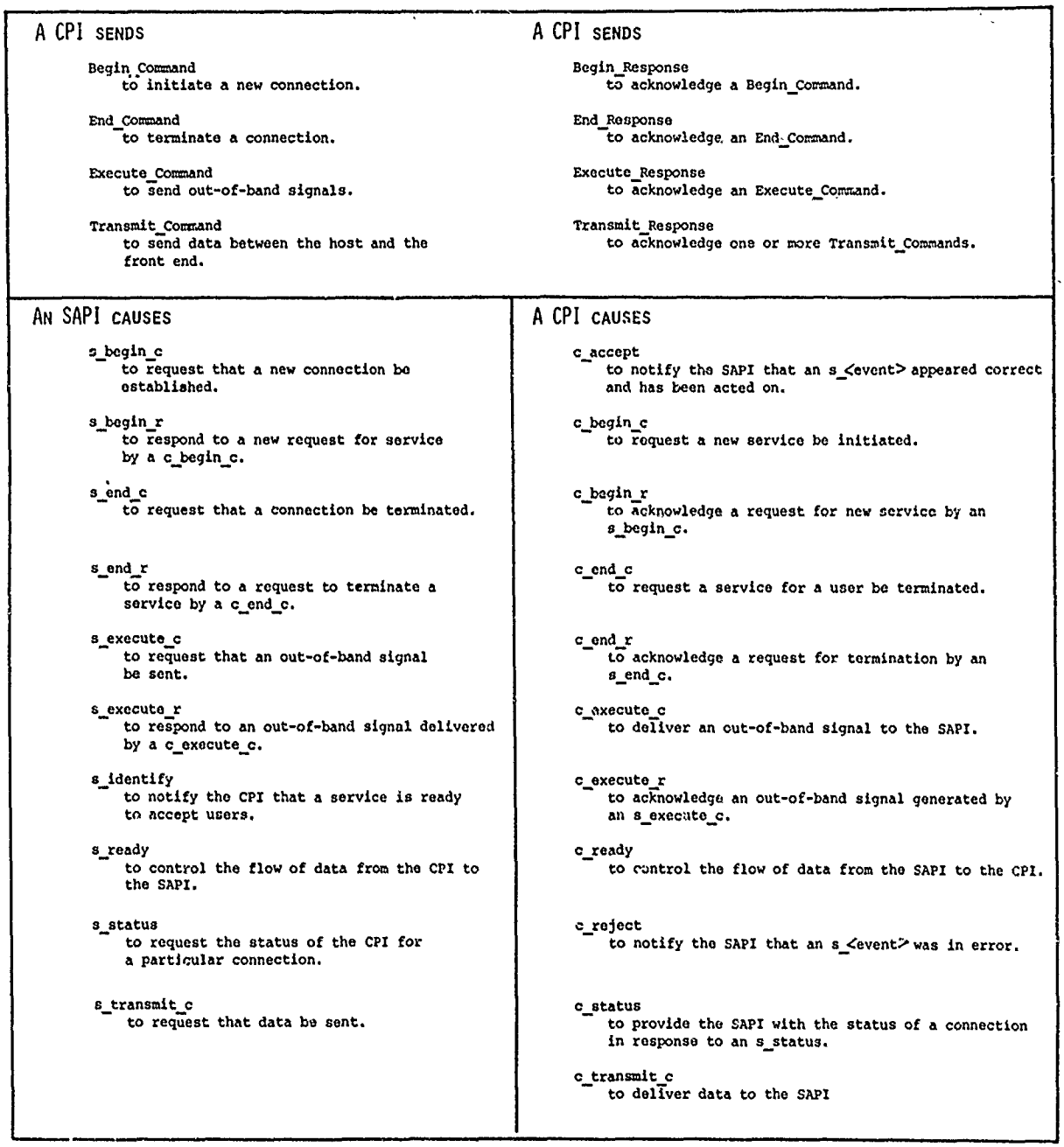
Note A: The size of PAD is an installation parameter.
Note B: The size of TEXT is computed by:
(Size) - (size of PAD) - 72.

Control	specifies control information for Execute_Commands and End_Commands.
Status	specifies status information in Responses.
PAD	is zero or more bits long and serves only to place TEXT on a convenient boundary.
TEXT	contains a service access or other higher level protocol message.

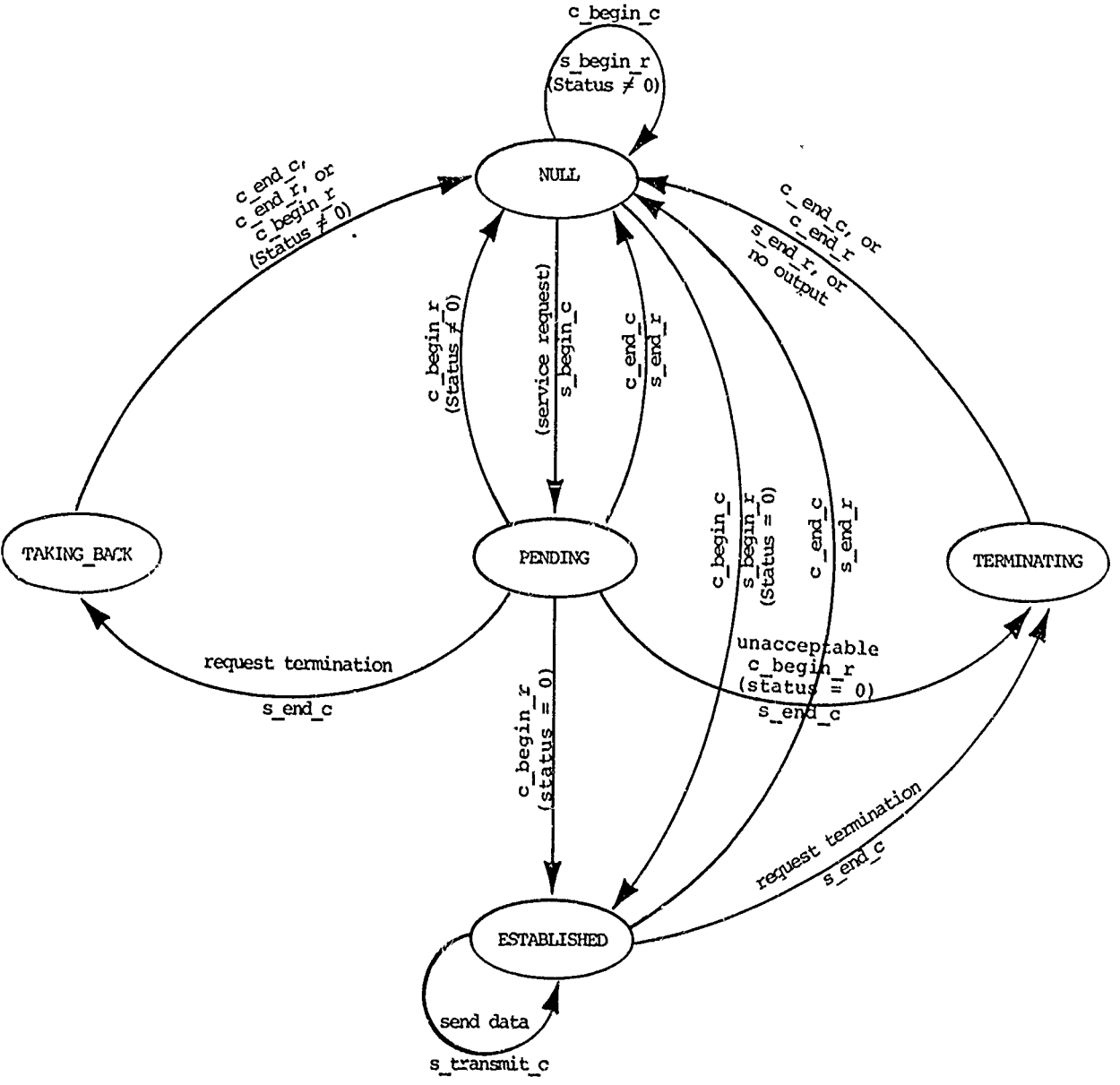
CHANNEL PROTOCOL HEADER



SYNOPTIC CHANNEL MACHINE STATE DIAGRAM



A Synopsis of Commands, Responses and Channel Interface Events



SYNOPTIC SAPI STATE DIAGRAM

Host Front End Protocol
Channel Machine
State Table

This section contains the detailed state transition table for CPI channel machines. There is a channel machine for each channel. A given channel machine receives inputs from and generates outputs for both the CPI multiplexor/demultiplexor (Commands and Responses) and an SAPI (events).

BLANK PAGE

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	acceptable s_begin_c	SENDER_ PENDING	c_accept Begin_Command	initialize channel machine
NULL	acceptable s_end_c (flushing, from HFP Maintenance Service)	- - - - -	c_accept End_Command (flushing)	
NULL	acceptable s_end_r (from HFP Maintenance Service)	- - - - -	c_accept End_Response	
NULL	acceptable s_identify	- - - - -	c_accept	register the SAPI
NULL	acceptable s_status	- - - - -	c_status	determine status
NULL	any other event (acceptable or unacceptable)	- - - - -	c_reject	log the error
NULL	valid Begin_Command	RECEIVER_ PENDING	c_begin_c	initialize channel machine
NULL	End_Command (valid or invalid, flushing or non-flushing)	- - - - -	End_Response	discard the message
NULL	End_Response	- - - - -	- - - - -	discard the message
NULL	any other Command (valid or invalid)	- - - - -	corresponding Response (Status=2)	log the error
NULL	any other Response (valid or invalid)	- - - - -	- - - - -	log the error

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
SENDER_ PENDING	acceptable s_end_c (flushing or non-flushing)	SENDER_ TAKING_BACK	c_accept End_Command (flushing)	
SENDER_ PENDING	acceptable s_status	- - - - -	c_status	determine status
SENDER_ PENDING	any other event (acceptable or unacceptable)	- - - - -	c_reject	log the error
SENDER_ PENDING	valid Begin_Response (Status=0)	ESTABLISHED	c_begin_r (Status=0)	
SENDER_ PENDING	valid Begin_Response (Status≠0)	NULL	c_begin_r (Status≠0)	- - - - -
SENDER_ PENDING	invalid Begin_Response	SENDER_ TERMINATING	c_begin_r (Status≠0) End_Command (flushing)	log the error
SENDER_ PENDING	End_Command (valid or invalid, flushing or non-flushing)	NULL	c_end_c End_Response	
SENDER_ PENDING	any other Command (valid or invalid)	- - - - -	corresponding Response (Status=2)	log the error
SENDER_ PENDING	any other Response (valid or invalid)	- - - - -	- - - - -	log the error

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
RECEIVER_ PENDING	acceptable s_begin_r (Status=Ø)	ESTABLISHED	c_accept Begin_Response (Status=Ø)	
RECEIVER_ PENDING	acceptable s_begin_r (Status≠Ø)	NULL	c_accept Begin_Response (Status≠Ø)	
RECEIVER_ PENDING	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	
RECEIVER_ PENDING	acceptable s_status	- - - - -	c_status	determine status
RECEIVER_ PENDING	any other event (acceptable or unacceptable)	- - - - -	c_reject	log the error
RECEIVER_ PENDING	valid End_Command	RECEIVER_ TAKING_BACK	c_end_c	
RECEIVER_ PENDING	invalid End_Command	NULL	c_end_c End_Response	log the error
RECEIVER_ PENDING	any other Command (valid or invalid)	- - - - -	corresponding Response (Status=2)	log the error
RECEIVER_ PENDING	any other Response (valid or invalid)	- - - - -	- - - - -	log the error

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
SENDER_ TAKING_BACK	acceptable s_end_c (flushing or non-flushing)	- - - - -	c_accept End_Command (flushing)	
SENDER_ TAKING_BACK	acceptable s_status	- - - - -	c_status	determine status
SENDER_ TAKING_BACK	any other event (acceptable or unacceptable)	- - - - -	c_reject	log the error
SENDER_ TAKING_BACK	valid Begin_Response (Status=0)	- - - - -	- - - - -	
SENDER_ TAKING_BACK	valid Begin_Response (Status≠0)	NULL	c_begin_r (Status≠0)	
SENDER_ TAKING_BACK	valid End_Command	NULL	c_end_c End_Response	
SENDER_ TAKING_BACK	valid End_Response	NULL	c_end_r	
SENDER_ TAKING_BACK	any other Command (valid or invalid)	- - - - -	send corresponding Response (Status=2)	log the error
SENDER_ TAKING_BACK	any other Response (valid or invalid)	- - - - -	- - - - -	log the error

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
RECEIVER_ TAKING_BACK	acceptable s_begin_r (Status=0)	- - - - -	c_reject	
RECEIVER_ TAKING_BACK	acceptable s_begin_r (Status≠0)	NULL	c_accept Begin_Response (Status≠0)	
RECEIVER_ TAKING_BACK	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	
RECEIVER_ TAKING_BACK	acceptable s_end_r	NULL	c_accept End_Response	
RECEIVER_ TAKING_BACK	acceptable s_status	- - - - -	c_status	determine status
RECEIVER_ TAKING_BACK	any other event (acceptable or unacceptable)	- - - - -	c_reject	log the error
RECEIVER_ TAKING_BACK	valid End_Command	NULL	c_end_c End_Response	
RECEIVER_ TAKING_BACK	any other Command (valid or invalid)	- - - - -	corresponding Response (Status=2)	log the error
RECEIVER_ TAKING_BACK	any other Response (valid or invalid)	RECEIVER_ TAKING_BACK	- - - - -	log the error

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	acceptable s_end_c (flushing)	SENDER_ TERMINATING	c_accept End_Command (flushing)	discard any queued data
ESTABLISHED (with data queued for apposite CPI)	acceptable s_end_c (non-flushing)	SENDER_ DRAINING	c_accept	discard any data queued for SAPI; send data to apposite CPI; place End_Command (non-flushing) at end of send queue; acknowledge but discard any incoming messages
ESTABLISHED (with no data queued for apposite CPI)	acceptable s_end_c (non-flushing)	SENDER_ TERMINATING	c_accept End_Command (non-flushing)	discard any data queued for SAPI; acknowledge but discard any incoming messages
ESTABLISHED	acceptable s_execute_c (synchronize)	- - - - -	c_accept Execute_Command	place Execute_Command at end of send queue
ESTABLISHED	acceptable s_execute_c (expedite)	- - - - -	c_accept Execute_Command	place Execute_Command at head of send queue
ESTABLISHED	acceptable s_execute_r	- - - - -	c_accept Execute_Response	place Execute_Response at head of send queue
ESTABLISHED (with no data queued for SAPI)	acceptable s_ready	- - - - -	c_accept	
ESTABLISHED (with data queued for SAPI)	acceptable s_ready	- - - - -	c_accept c_transmit_c(s)	
ESTABLISHED	acceptable s_status	- - - - -	c_status	determine status
ESTABLISHED	acceptable s_transmit_c	- - - - -	c_accept Transmit_Command	
ESTABLISHED	(able to accept data from SAPI or previous allocation exhausted)	- - - - -	c_ready	
ESTABLISHED	any other event (acceptable or unacceptable)	- - - - -	c_reject	log the error

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	valid End_Command (flushing)	RECEIVER_ TERMINATING	c_end_c	discard any data queued for SAPI and for apposite CPI
ESTABLISHED (with data queued for SAPI)	valid End_Command (non-flushing)	RECEIVER_ DRAINING	- - - - -	discard any data queued for apposite CPI; place c_end_c at end of receive queue; acknowledge but discard any incoming messages
ESTABLISHED (with no data queued for SAPI)	valid End_Command (non-flushing)	RECEIVER_ TERMINATING	c_end_c	discard any data queued for apposite CPI; acknowledge but discard any incoming messages
ESTABLISHED	invalid End_Command (flushing or non-flushing)	NULL	c_end_c End_Response	log the error
ESTABLISHED	valid Execute_Command (synchronize)	- - - - -	c_execute_c	place c_execute_c at end of receive queue
ESTABLISHED	valid Execute_Command (synchronize, attention)	- - - - -	c_execute_c	notify SAPI of attention, place c_execute_c at end of receive queue
ESTABLISHED	valid Execute_Command (expedite)	- - - - -	c_execute_c	place c_execute_c at head of receive queue
ESTABLISHED	valid Execute_Command (expedite, attention)	- - - - -	c_execute_c	notify SAPI of attention, place c_execute_c at head of receive queue
ESTABLISHED	valid Execute_Response	- - - - -	c_execute_r	place c_execute_r at head of receive queue
ESTABLISHED (with data queued for apposite CPI)	valid Transmit_Command	- - - - -	c_transmit_c Transmit_Command(s)	update flow control
ESTABLISHED (with no data queued for apposite CPI)	valid Transmit_Command	- - - - -	c_transmit_c Transmit_Response	update flow control
ESTABLISHED	invalid Transmit_Command	- - - - -	Transmit_Response (Status≠0)	log the error
ESTABLISHED	valid Transmit_Response	- - - - -	- - - -	update flow control; log any error
ESTABLISHED	any other Command (valid or invalid)	- - - - -	corresponding Response (Status=2)	log the error
ESTABLISHED	any other Response (valid or invalid)	- - - - -	- - - - -	log the error

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
SENDER_ DRAINING	acceptable s_end_c (flushing)	SENDER_ TERMINATING	c_accept End_Command (flushing)	discard any queued data
SENDER_ DRAINING	acceptable s_status	- - - - -	c_status	determine status
SENDER_ DRAINING	any other event (acceptable or unacceptable)	- - - - -	c_reject	log the error
SENDER_ DRAINING	valid End_Command (flushing)	NULL	c_end_c End_Response	discard any data queued
SENDER_ DRAINING	valid Execute_Command	- - - - -	Execute_Response (Status=39)	acknowledge but discard the message
SENDER_ DRAINING	valid Execute_Response	- - - - -	- - - - -	discard the message
SENDER_ DRAINING	valid Transmit_Command	- - - - -	Transmit_Response (Status=39)	acknowledge but discard any incoming messages
SENDER_ DRAINING	(acknowledgement of last Transmit_ Command	SENDER_ TERMINATING	End_Command (non-flushing)	
SENDER_ DRAINING	invalid Transmit_Command	- - - - -	Transmit Response	log the error
SENDER_ DRAINING	valid Transmit_Response	-	- - - -	update flow control
SENDER_ DRAINING	any other command (valid or invalid)	- - - - -	corresponding Response (Status=2)	log the error
SENDER_ DRAINING	any other Response (valid or invalid)	- - - - -	- - - - -	log the error

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
RECEIVER_ DRAINING	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	
RECEIVER_ DRAINING	acceptable s_execute_c	- - - - -		discard the message
RECEIVER_ DRAINING	acceptable s_execute_c		c_reject	discard the message
RECEIVER_ DRAINING (with data queued for send)	acceptable s_send_c	- - - - -	c_accept c_transmit_c	
RECEIVER_ DRAINING	acceptable s_status	- - - - -	c_status	determine status
RECEIVER_ DRAINING	acceptable s_transmit_c	- - - - -	c_reject	discard the message
RECEIVER_ DRAINING	(last c_transmit_c passed)	RECEIVER_ TERMINATING	c_end_c	
RECEIVER_ DRAINING	any other event (acceptable or unacceptable)	- - - - -	c_reject	log the error
RECEIVER_ DRAINING	valid End_Command (flushing)	RECEIVER_ TERMINATING	c_end_c	discard any data queued
RECEIVER_ DRAINING	any other Command (valid or invalid)	- - - - -	corresponding Response (Status=2)	log the error
RECEIVER_ DRAINING	any other Response (valid or invalid)	- - - - -	- - - - -	log the error

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
SENDER_ TERMINATING	acceptable s_end_c (flushing or non-flushing)	- - - - -	c_accept End_Command (flushing)	
SENDER_ TERMINATING	acceptable s_status	- - - - -	c_status	determine status
SENDER_ TERMINATING	any other event (acceptable or unacceptable)	- - - - -	c_reject	log the error
SENDER_ TERMINATING	valid End_Command (flushing or non-flushing)	NULL	c_end_c	
SENDER_ TERMINATING	invalid End_Command	NULL	c_end_c End_Response	log the error
SENDER_ TERMINATING	valid End_Response	NULL	c_end_r	
SENDER_ TERMINATING	valid Transmit_Command	- - - - -	- - - - -	discard the message
SENDER_ TERMINATING	valid Transmit_Response	- - - - -	- - - - -	update flow control
SENDER_ TERMINATING	any other Command (valid or invalid)	- - - - -	corresponding Response (Status=2)	log the error
SENDER_ TERMINATING	any other Response (valid or invalid)	- - - - -	- - - - -	log the error

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
RECEIVER_ TERMINATING	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	
RECEIVER_ TERMINATING	acceptable s_status	- - - - -	c_status	determine status
RECEIVER_ TERMINATING	acceptable s_end_r	NULL	c_accept End_Response	
RECEIVER_ TERMINATING	any other event (acceptable or unacceptable)	- - - - -	c_reject	log the error
RECEIVER_ TERMINATING	valid End_Command (flushing or non-flushing)	NULL	c_end_c End_Response	
RECEIVER_ TERMINATING	any other Command (valid or invalid)	- - - - -	corresponding Response (Status=2)	log the error
RECEIVER_ TERMINATING	any other Response (valid or invalid)	- - - - -	- - - - -	log the error

BLANK PAGE

CHANNEL INTERFACE

Introduction

The following section describes a model for the channel interface. The model is presented as a set of events corresponding to the Channel Protocol Commands and Responses, and a few events peculiar to the channel interface. It is of no concern here whether an implementation of the channel interface follows an event model or, say, a procedure calling model. However, the functions of the model must be preserved. For a key to the notation used in this section see "Notation and Nomenclature Conventions for the Channel Machine."

For each event, there is a presentation of:

1. its cause,
2. its effect,
3. the channel machine state table for it,
4. the semantics of its fields.

The conventions followed in the state tables are given in the introduction to the Complete Channel Machine State Table.

c_accept

Cause

A channel machine causes a c_accept event in order to indicate to the SAPI that an s_event appeared consistent and has been acted upon.

Effect

Any effect of a c_accept event on the SAPI depends heavily on the implementation.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
any state	acceptable s_<event>	(see the s_<event>)	c_accept (see the s_<event>)	(see the s_e<event>)

Syntax

```
Ref:        FIXED(?)
Group:      FIXED(12)
Member:     FIXED(16)
```

Semantics

Ref: specifies the unique identifier supplied by the SAPI to identify this response to a previous s_<event>.

Group and Member: specify the channel to which this event applies.

c_begin_c

Cause

A channel machine causes a c_begin_c event in order to request an SAPI to initiate access to a service.

Effect

When an SAPI receives a c_begin_c event, it determines whether or not it can initiate access to the service and then causes an s_begin_r event indicating the result.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	valid Begin_Command	RECEIVER_ PENDING	c_begin_c	initialize channel machine

Syntax

```
Group:          FIXED(12)
Member:         FIXED(16)
Service:        FIXED(16)
Text:           VARIABLE
```

Semantics

Group and Member: specifies the channel to be established.

Service: specifies the number of the SAPI to which the channel is to be established.

Text: contains the service access protocol message.

c_begin_r

Cause

A channel machine causes a c_begin_r event in order to notify the SAPI that it has received a Begin_Response.

Effect

When an SAPI receives a c_begin_r event with Status = Ø, it may proceed to exchange data with its apposite. If an SAPI receives a c_begin_r event with Status ≠ Ø, it enters the NULL state.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
SENDER_ PENDING	valid Begin_Response (Status=Ø)	ESTABLISHED	c_begin_r (Status=Ø)	
SENDER_ PENDING	valid Begin_Response (Status≠Ø)	NULL	c_begin_r (Status≠Ø)	- - - - -
SENDER_ PENDING	invalid Begin_Response	SENDER_ TERMINATING	c_begin_r (Status≠Ø) End_Command (flushing)	log the error
SENDER_ TAKING_BACK	valid Begin_Response (Status≠Ø)	NULL	c_begin_r (Status≠Ø)	

Syntax

```
Group:          FIXED(12)
Member:         FIXED(16)
Status:         FIXED(8)
Text:           VARIABLE
```

Semantics

Group and Member: specify the channel refered to in the Begin_Response.

Status: indicates to the SAPI whether the attempt to establish a channel has been successful. This field will contain one of the standard Status codes (see Complete Status Codes).

Text: contains the service access protocol message.

c_end_c

Cause

A channel machine causes a c_end_c event in order to pass the TEXT field of an End_Command to the SAPI.

Effect

When an SAPI receives a c_end_c event, it is to immediately terminate the access to the service and cause an s_end_r event.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
SENDER_ PENDING	End_Command (valid or invalid, flushing or non-flushing)	NULL	c_end_c End_Response	
RECEIVER_ PENDING	valid End_Command	RECEIVER_ TAKING_BACK	c_end_c	
RECEIVER_ PENDING	invalid End_Command	NULL	c_end_c End_Response	log the error
SENDER_ TAKING_BACK	valid End_Command	NULL	c_end_c End_Response	
RECEIVER_ TAKING_BACK	valid End_Command	NULL	c_end_c End_Response	
ESTABLISHED	valid End_Command (flushing)	[illegible]	[illegible]	discard any data queued for SAPI and for apposite CPI
ESTABLISHED (with data queued for SAPI)	[illegible]	RECEIVER_ DRAINING	- - - - -	discard any data queued for apposite CPI; place c_end_c at end of receive queue; acknowledge but discard any incoming messages
ESTABLISHED (with no data queued for SAPI)	valid End_Command (non-flushing)	RECEIVER_ TERMINATING	c_end_c	discard any data queued for apposite CPI; acknowledge but discard any incoming messages
ESTABLISHED	invalid End_Command (flushing or non-flushing)	NULL	c_end_c End_Response	log the error
SENDER_ DRAINING	valid End_Command (flushing)	NULL	c_end_c End_Response	discard any data queued
RECEIVER_ DRAINING	(last c_transmit_c passed)	RECEIVER_ TERMINATING	c_end_c	
RECEIVER_ DRAINING	valid End_Command (flushing)	RECEIVER_ TERMINATING	c_end_c	discard any data queued
SENDER_ TERMINATING	valid End_Command (flushing or non-flushing)	NULL	c_end_c	
SENDER_ TERMINATING	invalid End_Command	NULL	c_end_c End_Response	log the error
RECEIVER_ TERMINATING	valid End_Command (flushing or non-flushing)	NULL	c_end_c End_Response	

Syntax

Group:	**FIXED(15)**
Member:	**FIXED(15)**
Status:	**FIXED(8)**
Text:	**VARIABLE**

Semantics

Group and Member: specify the channel to be terminated. If the Group is not zero and the Member is zero, all channels with the same group are to be terminated. If both Group and Member are zero, all cnannels are to be terminated.

Status: indicates to the service whether or not the End_Command was correct. This field will contain one of the standard Status codes (see c_reject).

Text: contains the service access protocol message.

c_end_r

Cause

A channel machine causes a c_end_r event in order to pass the TEXT field of an End_Response to the SAPI and to indicate to it that the channel has been terminated.

Effect

When a SAPI receives a c_end_r event, it considers the channel to be terminated.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
SENDER_ TAKING_BACK	valid End_Response	NULL	c_end_r	
SENDER_ TERMINATING	valid End_Response	NULL	c_end_r	

Syntax

```
Group:      FIXED(12)
Member:     FIXED(16)
Status:     FIXED(8)
Text:       VARIABLE
```

Semantics

Group and Member: specify the channel on which an End_Response was received.

Status: indicates to the service whether or not the End_Response was correct. This field will contain one of the standard Status codes (see c_reject).

Text: contains the service access protocol message.

c_execute_c

Cause

A channel machine causes a c_execute_c event in order to pass the TEXT field of an Execute_Command to the SAPI.

Effect

When an SAPI receives a c_execute_c event, it is to act on the TEXT of the Execute_Command immediately and cause an s_execute_r event.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	valid Execute_Command (synchronize)	- - - - -	c_execute_c	place c_execute_c at end of receive queue
ESTABLISHED	valid Execute_Command (synchronize, attention)	- - - - -	c_execute_c	notify SAPI of attention, place c_execute_c at end of receive queue
ESTABLISHED	valid Execute_Command (expedite)	- - - - -	c_execute_c	place c_execute_c at head of receive queue
ESTABLISHED	valid Execute_Command (expedite, attention)	- - - - -	c_execute_c	notify SAPI of attention, place c_execute_c at head of receive queue

Syntax

Group: FIXED(12)
Member: FIXED(16)
Text: VARIABLE

Semantics

Group and Member: specify the channel on which an Execute_Command was received.

Text: contains the service access protocol message.

c_execute_r

Cause

A channel machine causes a c_execute_r event in order to pass the TEXT field of an Execute_Response to the SAPI.

Effect

When an SAPI receives a c_execute_r event, it is to process the Text field.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	valid Execute_Response	- - - - -	c_execute_r	place c_execute_r at head of receive queue

Syntax

```
Group:      FIXED(12)
Member:     FIXED(16)
Status:     FIXED(8)
Text:       VARIABLE
```

Semantics

Group and Member: specify the channel on which an Execute_Response has been received.

Status: indicates to the SAPI whether or not the Execute_Command was correctly formulated and any requested action has completed. The field will contain one of the standard Status codes (see c_reject).

Text: contains the service access protocol message.

c_ready

Cause

A channel machine causes a c_ready event in order to notify the SAPI that it is able to accept data from it.

Effect

If the SAPI has data queued for the channel machine, it attempts to transfer the data to it via an s_transmit_c event, and, if necessary, updates the channel interface flow control via an s_ready event.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	(able to accept data from SAPI or previous allocation exhausted)	- - - - -	c_ready	

Syntax

```
Group:          FIXED(12)
Member:         FIXED(16)
Msgs:           FIXED(8)
```

Semantics

Group and Member: specify the channel to which this event applies.

Msgs: specifies the number of Transmit_Commands the channel machine is currently able to accept. The value of Msgs replaces any previous value.

c_reject

Cause

A channel machine causes a c_reject event in order to notify the SAPI that an s_event appeared inconsistent and has not been acted upon.

Effect

When an SAPI receives a c_reject event, it enters an error recovery phase.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
any state	(any unacceptable s_<event>)	- - - - -	c_reject	log the error

Syntax

```
Ref:        FIXED(?)
Group:      FIXED(12)
Member:     FIXED(16)
Reason:     FIXED(8)
```

Semantics

Ref: specifies the unique identifier supplied by the SAPI to identify this response to a previous s_<event>.

Group and Member: specify the channel to which this event applies.

Reason: specifies a code indicating why the event referred to in the Ref field is being rejected. The reasons for rejecting an event are assigned the following codes:

Reason	Meaning
1	Channel non-existent: the Group and Member fields of an s_event (other than an s_begin_c or an s_execute_c event) referenced a virtual channel unknown to the receiver.
2	Illegal state: an s_event was received referencing a channel which was in a state for which the event type was illegal.
3	Event not implemented: an s_event was received whose type was legal but not implemented. Currently this can only be an s_begin_c, s_execute_c, s_execute_r, or s_status events.
5	Field too long: the length of a field in the s_event exceeded the maximum size permitted at the installation.
6	Not used.
7	Illegal field value or combination of values: the values of a field in the s_event referred to by the Ref code contained an illegal field value or a field value that was inconsistent with other field values specified by the s_event.
8	Illegal type: the s_event was of a type unknown to the CPI.
9	Illegal Group and Member: the s_event specified a Group and Member that is not accessible to this SAPI.
10	Group(s) terminating: communication is being terminated for this Group or for all Groups.
32	Channel in use: the channel referenced in the s_begin_c event was already assigned (i.e., not in the NULL state).
33	Reserved.
34	Reserved.
35	Reserved.

36 No allocation: an s_transmit_c event has been received and discarded because the SAPI has exhausted the flow control allocation given to it via a c_ready event.

37 Bad channel polarity: the high-order bit of the Group field in the s_begin_c had the wrong value.

38 Channel down: HFP communication is not enabled.

39 Command discarded: the channel machine received a Transmit_Command or an Execute_Command, is in the SENDER_DRAINING or SENDER_TERMINATING state, and has discarded the command without passing it to the service access level.

c_status

Cause

A channel machine causes a c_status event in order to notify the SAPI of the its status.

Effect

The effect of a c_status event on the SAPI will depend heavily on the installation.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
any state	acceptable s_status	- - - - -	c_status	determine status

Syntax

```
Group:          FIXED(12)
Member:         FIXED(16)
State:          FIXED(8)
Snd_Msgs:       FIXED(8)
Rcv_Msgs:       FIXED(8)
```

Semantics

Group and Member: specify the channel to which this event applies.

State: contains a value indicating the current state of the channel machine. The encoded values of the states are:

```
Ø     NULL
1     SENDER_PENDING
```

2	RECEIVER_PENDING
3	SENDER_TAKING_BACK
4	RECEIVER_TAKING_BACK
5	ESTABLISHED
6	SENDER_DRAINING
7	RECEIVER_DRAINING
8	SENDER_TERMINATING
9	RECEIVER_TERMINATING

Snd Msgs: contains the number of events the channel machine can pass to the SAPI within the limits of the channel interface flow control.

Rcv Msgs: contains the number of events the channel machine is able to accept.

c_transmit_c

Cause

A channel machine causes a c_transmit_c event in order to pass the TEXT field of a Transmit_Command to the SAPI.

Effect

When a SAPI receives a c_transmit_c event, it processes the data transferred from the channel machine and, if necessary, updates the channel interface flow-control via an s_ready event.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED (with data queued for SAPI)	acceptable s_ready	- - - - -	c_accept; c_transmit_c(s)	
ESTABLISHED (with data queued for apposite CPI)	valid Transmit_Command	- - - - -	c_transmit_c Transmit_Command(s)	update flow control
ESTABLISHED (with no data queued for apposite CPI)	valid Transmit_Command	- - - - -	c_transmit_c Transmit_Response(s)	update flow control
RECEIVER_ DRAINING (with data queued for SAPI)	acceptable s_ready	- - - - -	c_accept c_transmit_c	
RECEIVER_ DRAINING	(last c_transmit_c passed)	RECEIVER_ TERMINATING	c_end_c	

Syntax

Group:	FIXED(12)
Member:	FIXED(16)
Control:	FIXED(8)
Text:	VARIABLE

Semantics

Group and Member: specify the virtual channel on which a Transmit_Command has been received.

Control: undefined.

Text: contains the service access protocol message.

s_begin_c

Cause

An SAPI causes an s_begin_c event in order to request its CPI to establish a host to front end channel.

Effect

If the CPI detects an inconsistency in the s_begin_c event, it causes a c_reject event. If no inconsistency is detected, a channel machine is initialized which then sends a corresponding Begin_Command.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	acceptable s_begin_c	SENDER_ PENDING	c_accept Begin_Command	initialize channel machine
any other state	s_begin_c (acceptable or unacceptable)	- - - - -	c_reject	log the error

Syntax

```
Ref:        FIXED(?)
Group:      FIXED(12)
Member:     FIXED(16)
Service:    FIXED(16)
Text:       VARIABLE
```

Semantics

Ref: specifies a unique identifier supplied by the SAPI to identify the response to this event via the c_accept or c_reject events.

Group and Member: specifies the virtual channel to be established. If this field is zero, the CPI will choose the Group value.

Service: specifies the number of the SAPI to which the virtual channel is to be established.

Text: contains the service access protocol message.

s_begin_r

Cause

An SAPI causes an s_begin_r event in order to respond to a c_begin_c event.

Effect

If the CPI detects an inconsistency in the s_begin_r event, it causes a c_reject event. If no inconsistency is detected, the channel machine then sends a corresponding Begin_Response.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
RECEIVER_ PENDING	acceptable s_begin_r (Status=0)	ESTABLISHED	c_accept Begin_Response (Status=0)	
RECEIVER_ PENDING	acceptable s_begin_r (Status≠0)	NULL	c_accept Begin_Response (Status≠0)	
RECEIVER_ TAKING_BACK	acceptable s_begin_r (Status=0)		c_reject	
RECEIVER_ TAKING_BACK	acceptable s_begin_r (Status≠0)	NULL	c_accept Begin_Response (Status≠0)	
any other state	s_begin_r (acceptable or unacceptable)	- - - - -	c_reject	log the error

Syntax

```
Ref:        FIXED(?)
Group:      FIXED(12)
Member:     FIXED(16)
```

Status:	FIXED(8)
Text:	VARIABLE

Semantics

Ref: specifies a unique identifier supplied by the SAPI to identify the response to this event via the c_accept or c_reject events.

Group and Member: specify the channel referred to by the c_begin_c event.

Status: indicates to the channel machine whether the attempt to establish a channel has been successful. This field will contain one of the standard status codes (see c_reject).

Text: contains the service access protocol message.

s_end_c

Cause

An SAPI causes an s_end_c event in order to terminate access to the service.

Effect

If the CPI detects an inconsistency in the s_end_c event, it causes a c_reject event. If no inconsistency is detected, the channel machine then sends a corresponding End_Command.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	acceptable s_end_c (flushing, from HFP Maintenance Service)	- - - - -	c_accept End_Command (flushing)	
SENDER_ PENDING	acceptable s_end_c (flushing or non-flushing)	SENDER_ TAKING_BACK	c_accept End_Command (flushing)	
RECEIVER_ PENDING	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	
SENDER_ TAKING_BACK	acceptable s_end_c (flushing or non-flushing)	- - - - -	c_accept End_Command (flushing)	
RECEIVER_ TAKING_BACK	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	
ESTABLISHED	acceptable s_end_c (flushing)	SENDER_ TERMINATING	c_accept End_Command (flushing)	discard any queued data
ESTABLISHED (with data queued for apposite CPI)	acceptable s_end_c (non-flushing)	SENDER_ DRAINING	c_accept	discard any data queued for SAPI; send data to apposite CPI; place End_Command (non-flushing) at end of send queue; acknowledge but discard any incoming messages
ESTABLISHED (with no data queued for apposite CPI)	acceptable s_end_c (non-flushing)	SENDER_ TERMINATING	c_accept End_Command (non-flushing)	discard any data queued for SAPI; acknowledge but discard any incoming messages
SENDER_ DRAINING	acceptable s_end_c (flushing)	SENDER_ TERMINATING	c_accept End_Command (flushing)	discard any queued data
RECEIVER_ DRAINING	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	discard any data queued
SENDER_ TERMINATING	acceptable s_end_c (flushing or non-flushing)	- - - - -	c_accept End_Command (flushing)	
RECEIVER_ TERMINATING	acceptable s_end_c (flushing or non-flushing)	NULL	c_accept End_Command (flushing)	
any other state	s_end_c (acceptable or unacceptable, flushing or non-flushing)	- - - - -	c_reject	

Syntax

Ref:	FIXED(?)
Group:	FIXED(12)
Member:	FIXED(16)
Control:	FIXED(8)
Text:	VARIABLE

Semantics

Ref: specifies a unique identifier supplied by the SAPI to identify the response to this event via the c_accept or c_reject events.

Group and Member: specify the channel to be terminated. If the Group is not zero and the Member is zero, all channels with the same group are to be terminated. If both Group and Member are zero, all virtual channels are to be terminated.

Control: The Control field bit has the following meaning:

Flush away: (bit 1 = 1) means immediately flush data which is queued going away from the sender of the End_Command and terminate the channel. If this option is not requested (i.e., if bit 1 = Ø), the End_Command is not to take effect until all data queued going away from the sender of the End_Command has been processed in the normal manner.

Text: contains the service access protocol message.

s_end_r

Cause

An SAPI causes an s_end_r event in order to respond to a c_end_c event.

Effect

If the CPI detects an inconsistency in the s_end_r event, it causes a c_reject event. If no inconsistency is detected, the channel machine then sends a corresponding End_Response.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	acceptable s_end_r (from EFP Maintenance Service)	- - - - -	c_accept End_Response	
RECEIVER_ TAKING_DATA	acceptable s_end_r	NULL	c_accept End_Response	
RECEIVER_ TERMINATING	acceptable s_end_r	NULL	c_accept End_Response	
any other state	s_end_r (acceptable or unacceptable)	- - - - -	c_reject	

Syntax

Ref:	FIXED(?)
Group:	FIXED(12)
Member:	FIXED(16)
Status:	FIXED(8)
Text:	VARIABLE

Semantics

Ref: specifies a unique identifier supplied by the SAPI to identify the response to this event via the c_accept or c_reject events.

Group and Member: specify the virtual channel referenced in the c_end_c event to which this is a response.

Status: indicates to the channel machine whether or not the c_end_c event has been successfully completed. This field will contain one of the standard Status codes (see c_reject).

Text: contains the service access protocol message.

s_execute_c

Cause

An SAPI causes an s_execute_c event in order to send data to its apposite outside the normal flow of data.

Effect

If the CPI detects an inconsistency in the s_execute_c event, it causes a c_reject event. If no inconsistency is detected, the channel machine then sends a corresponding Execute_Command.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	acceptable s_execute_c (synchronize)	- - - - -	c_accept Execute_Command	place Execute_Command at end of send queue
ESTABLISHED	acceptable s_execute_c (expedite)	- - - - -	c_accept Execute_Command	place Execute_Command at head of send queue
RECEIVER_ DRAINING	acceptable s_execute_c	- - - - -	c_reject	discard the message
any other state	s_execute_c (acceptable or unacceptable)	- - - - -	c_reject	log the error

Syntax

```
Ref:        FIXED(?)
Group:      FIXED(12)
Member:     FIXED(16)
Control:    FIXED(8)
Text:       VARIABLE
```

Semantics

Ref: specifies a unique identifier supplied by the SAPI to identify the response to this event via the c_accept or c_reject events.

Group and Member: specify the channel to which this event applies.

Control: The Control field bits have the following meaning:

Attention: (bit Ø = 1) means inform the SAPI module of the arrival of an Execute_Command with the Attention bit set.

Synchronize: (bit 3 = 1) means place the Execute_Command at the end of the Sending Queue of the local CPI.

The Execute_Command is executed at the remote CPI as follows:

If synchronize,

place the Execute_Command at the end of the queue of data to be read by the SAPI module;

otherwise,

place the Execute_Command at the beginning of the queue of data to be read by the the SAPI module.

if attention,

immediately notify the SAPI module of the arrival of the Execute_Command.

The meanings of the various bit value combinations to the remote CPI are summarized in Table 1.

Table 1

EXECUTE Command Control Field Bit Values

Bit Ø123	Meaning
ØØØØ	Place the Execute_Command at the head of the Receiving Queue.
1ØØØ	Place the Execute_Command at the head of the Receiving Queueueue and notify the SAPI module of the attention.

0001	Place the Execute_Command at the end of the Receiving Queue.
1001	Place the Execute_Command at the end of the Receiving Queue. Notify the SAPI module of the attention.

Text: contains the service access protocol message.

s_execute_r

Cause

An SAPI causes an s_execute_r event in order to respond to a c_execute_c event.

Effect

If the CPI detects an inconsistency in the s_execute_r event, it causes a c_reject event. If no inconsistency is detected, the channel machine then sends a corresponding Execute_Response.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	acceptable s_execute_r	- - - - -	c_accept Execute_Response	place Execute_Response at head of send queue
RECEIVER_DRAINING	acceptable s_execute_r	- - - - -	c_reject	discard the message
any other state	s_execute_r (acceptable or unacceptable)	- - - - -	c_reject	log the error

Syntax

```
Ref:        FIXED(?)
Group:      FIXED(12)
Member:     FIXED(16)
Status:     FIXED(8)
Text:       VARIABLE
```

Semantics

Ref: specifies a unique identifier supplied by the SAPI to identify the response to this event via the c_accept or c_reject events.

Group and Member: specify the channel to which this event applies.

Status: indicates to the channel machine whether or not the Execute_Command was correctly formulated and any requested action has completed. The field will contain one of the standard Status codes (see c_reject).

Text: contains the service access protocol message.

s_identify

Cause

An SAPI causes an s_identify event in order to identify itself to its CPI.

Effect

When a CPI receives an s_identify event, it attempts to validate the SAPI. The attempt may fail if the SAPI already exists or if it is not privileged or if an inconsistency in the s_identify event is detected. In this case, the CPI will cause a c_reject event. If the CPI is able to validate the SAPI, it will cause a c_accept event.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	acceptable s_identify	- - - - -	c_accept	register the SAPI
any other state	s_identify (acceptable or unacceptable)	- - - - -	c_reject	log the error

Syntax

Ref: FIXED(?)
Service: FIXED(16)

Semantics

Ref: specifies a unique identifier supplied by the SAPI so that it can identify the eventual c_accept event or c_reject event.

Service: specifies the number of the SAPI which is to be validated with the CPI.

Note: To provide protection against a process masquerading as an SAPI, some sites may require additional fields for validating an SAPI.

s_ready

Cause

An SAPI causes an s_ready event in order to notify a channel machine that it is able to accept data.

Effect

If the CPI detects an inconsistency in the s_ready event, it causes a c_reject event. If no inconsistency is detected, then if the channel machine has data queued for the SAPI, it attempts to transfer the data to the SAPI via a c_transmit_c event and updates the channel interface flow-control.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED (with no data queued for SAPI)	acceptable s_ready	- - - - -	c_accept	
ESTABLISHED (with data queued for SAPI)	acceptable s_ready	- - - - -	c_accept c_transmit_c(s)	
RECEIVER_ DRAINING (with data queued for SAPI)	acceptable s_ready	- - - - -	c_accept c_transmit_c	
any other state	s_ready (acceptable or unacceptable)	- - - - -	c_reject	log the error

Syntax

```
Ref:        FIXED(?)
Group:      FIXED(12)
Member:     FIXED(16)
Msgs:       FIXED(8)
```

Semantics

Ref: specifies a unique identifier supplied by the SAPI to identify the response to this event via the c_accept or c_reject events.

Group and Member: specify the channel to which this event applies.

Msgs: specifies the number of Transmit_Commands the SAPI is able to accept on this channel. The value of Msgs replaces any previous value.

s_status

Cause

An SAPI causes an s_status event in order to obtain information about the status of a channel.

Effect

When a channel machine receives an s_status event, it is to determine its present state and cause a c_status event to notify the SAPI.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
any state	acceptable s_status	- - - - -	c_status	determine status

Syntax

```
Ref:        FIXED(?)
Group:      FIXED(12)
Member:     FIXED(16)
```

Semantics

Ref: contains a unique identifier supplied by the SAPI to identify the response to this event via the c_accept or c_reject events.

Group and Member: specify the channel on which status has been requested.

s_transmit_c

Cause

An SAPI causes an s_transmit_c event in order to send flow-controlled, in-order data to its apposite.

Effect

If the CPI detects an inconsistency in the s_transmit_c event, it causes a c_reject event. If no inconsistency is detected, the channel machine then sends a corresponding Transmit_Command. If the channel machine determines that the channel interface flow control should be updated, it also causes a c_ready event.

Channel Machine States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	acceptable s_transmit_c	- - - - -	c_accept Transmit_Command	
RECEIVER_ DRAINING	acceptable s_transmit_c	- - - - -	c_reject	discard the message
any other state	s_transmit_c (acceptable or unacceptable)	- - - - -	c_reject	log the error

Syntax

Ref:	FIXED(?)
Group:	FIXED(12)
Member:	FIXED(16)
Control:	FIXED(8)
Text:	VARIABLE

Semantics

Ref: specifies a unique identifier supplied by the SAPI to identify the response to this event via the c_accept or c_reject events.

Group and Member: specify the channel to which this event applies.

Control: is undefined.

Text: contains the service access protocol message.

INITIALIZING HOST - FRONT END COMMUNICATION

To initialize (or restart) host - front end communication, the following sequence of steps must be taken:

HOST: 1) The host HFP Maintenance Service causes an s_end_c event with Group=Member=Ø.

HOST: 2) The host CPI causes a flushing c_end_c event for each channel and sends a flushing End_Command with Group = Member = Ø to the front end CPI.

FRONT END: 3) The front end CPI causes a flushing c_end_c event for each channel and sends an End_Response with Group = Member = Ø to the host CPI.

HOST: 4) The host CPI causes a c_end_r event for the host HFP Maintenance Service to indicate that all connections have been terminated.

BOTH: 5) If necessary, the host and the front end reinitialize link level communication. How this is done is installation dependent.

HOST: 6) The host HFP Maintenance Service causes an s_begin_c event.

HOST: 7) The host CPI sends a Begin_Command with Service code = <HFP Maintenance Service Code>.

FRONT END: 8) The front end CPI causes a c_begin_c event for the HFP Maintenance Service.

FRONT END: 9) The front end HFP Maintenance Service causes an s_begin_r with Status = Ø.

FRONT END: 1Ø) The front end CPI sends a Begin_Response with Status = Ø.

HOST: 11) The host CPI causes a c_begin_r event with Status = Ø for the HFP Maintenance Service.

When the above steps have been completed, host - front end communication may proceed (i.e., virtual channels may be established).

BLANK PAGE

SPECIFYING SERVICE ACCESS PROTOCOLS

Introduction

Together, the link and channel protocols provide a reliable virtual communications medium for the service access protocols. Each service access protocol provides the host with access to a specific service offloaded to the front end, e.g., a network data transport service or a network virtual terminal (for a detailed discussion, see Day, J.; Offloading ARPANET Protocols to a Front End, CAC Document No. 230, 1977). Since each service access protocol definition depends on the service to which it provides access, the service access protocols cannot be specified in the present document. However, to ensure uniformity, consistency, and completeness, the forms that service access protocol specifications and adaptation descriptions (see below) should follow have been specified.

Specifications and Adaptation Descriptions

Each service access protocol is described by

a) a service access protocol specification, and

b) a set of service access protocol adaptation descriptions.

A service access protocol specification defines the rules for

communication between apposite SAPIs in the host and in the front end. The unit of service access protocol communication is the service access protocol message. Service access protocol messages correspond to channel protocol messages, i.e., there is a service access protocol begin_command corresponding to the channel protocol Begin_Command, a service access protocol transmit_command corresponding to the channel protocol Transmit_command, etc. (To distinguish service access protocol messages from channel protocol messages, service access protocol messages are written all lower-case.) Service access protocol messages are carried by the TEXT field of the corresponding channel protocol messages. Thus, specifying a service access protocol amounts to defining the TEXT fields of the channel protocol messages in terms of the corresponding service access protocol messages.

Since choices must be made in implementing a service access protocol for a given host and front end (e.g., in handling mismatch between host and front end word sizes and/or character sets), an adaptation description describing these choices must also be made for each implementation of the service access protocol.

Service Access Protocol Specifications

Each service access protocol specification shall have the following form:

<name of the service to which access is provided>
Service Access Protocol
Specification

I. HFP Code Number for the <to be accessed> Service:

Give the HFP Code Number for the service to be accessed.

II. Description of the Service to be Accessed:

Describe the service to be accessed. The description of the service is to include:

a) its purpose,

b) background information,

c) an overview of its operation,

d) optional features,

e) supporting services that must also be implemented, and

f) an overview of the offloading strategies relevant to this service,

g) references to relevant documents.

III. Message Use

Describe how each service access protocol message is used. The messages are to be discussed in the order:

begin_command	(Section III.A)
begin_response	(Section III.B)
end_command	(Section III.C)
end_response	(Section III.D)

execute_command	(Section III.E)
execute_response	(Section III.F)
transmit_command	(Section III.G)

Note: the TEXT field of a Transmit_Response is never passed to an SAPI by a CPI. Therefore, there is no service access protocol transmit_response corresponding to the channel protocol Transmit_Response.

The description of each message is to have the following form:

III.<X>.1 When Sent

Describe the circumstances under which this message is sent.

III.<X>.2 Action on Receipt

Describe the actions to be taken by the receiver of this message.

III.<X>.3 Syntax

Define the syntax of the service access protocol message. Include the minimum length of each field. For the detailed format of this section, see "Specifying Fields" below.

III.<X>.4 Semantics

Define the semantics of each field defined in III.<X>.3 "Syntax". Each definition will include at least:

a) the meaning of each value for each field,

b) the restrictions on the values for each field,

c) the default value for each field and how it is expressed, and

d) the interaction of the values of each field with the meanings of other fields.

Specifying Fields

Service access protocol messages are carried by the TEXT field of their corresponding channel protocol messages. TEXT in channel protocol messages consists of fields. The value of each field represents a datum. The datum may be simple, e.g., a network port number. Or the datum may be a complex of data, e.g., the set of parameters necessary for initiating a network connection.

The data represented by a field may always require the same field length, e.g., ARPANET socket numbers always require 32 bits. A field representing such fixed-length data is called a fixed field. It consists of a bit string (at least) long enough to represent the data.

The data represented by a field may require varying field lengths, e.g., user names consisting of character strings. A field representing such variable-length data is called a variable field. It consists of a fixed length count part followed by a bit string content part. The content part must be long enough to represent the datum. The value of the count part is the length of the content part. The length of the count part must be chosen to be large enough to represent the length of the largest conceivable content part. This length may be expressed in bits, characters, bytes, words, or any other convenient units. The units must be specified together with each field in the Syntax section (if known when the service access protocol is specified) and in the adaptation description.

The data represented by a variable field may be complex, and the fields representing the values of the data may themselves be fixed or variable, simple or complex. In some cases, it may be desirable to have a fixed qualifier field as the first field representing a complex datum. This qualifier field will aid in the parsing and interpretation of the rest of the fields.

A service access protocol specification describes the features of the service access protocol common to all implementations. Word sizes and convenient data alignment boundaries may differ among implementations. Therefore, a service access protocol specification cannot prescribe the field formats for all implementations. But a service access protocol specification can and must specify what fields are to be present, their content, and their minimum lengths.

A fixed field is to be represented by:

<field name> : FIXED(<length>)

A variable field is to be represented by:

<field name> : VARIABLE(<count part length>)

A complex field is to be represented by:

```
<field name> : COMPLEX(<count part length>)
    <field name> : .........
          .        .        .
          .        .        .
          .        .        .
    <field name> : .........
```

where hierarchy of structure is indicated by indentation.

Defaults

The default value is expressed by:

a) the absence of the field, if it is part of a complex field and all subsequent fields which are part of the complex field are also absent;

b) zero length, if the field is a variable field;

c) a predetermined value (e.g., zero), if the field is a fixed field which does not meet condition a).

Adaptation Descriptions

Each service access protocol adaptation description shall have the following form:

<name of the service>
Service Access Protocol
Adaptation Description

Section I. Description of the Adaptation

I.A Service Access Protocol

Give the complete reference citation for the service access protocol specification on which this adaptation is based, including document numbers, date, etc.

I.B Applicable Hosts and Front Ends

Present a table showing the classes of hosts and front ends for which this adaptation is intended.

I.C Supported Facilities List

Present a table showing the optional facilities supported by this adaptation. A more detailed discussion of each facility supported is to be found in section II.B.

I.D Unsupported Facilities List

Present a table showing the optional facilities not supported by this adaptation. A more detailed discussion of the limitations of this adaptation is to be given in section II.C.

Section II. Discussion of the Adaptation

II.A Distribution of Functions

Discuss the scheme or schemes supported by this adaptation for distributing functions between the host and the front end.

II.B Optional Facilities Supported

Discuss each of the facilities listed in section I.C. This section is to contain a subsection for

each of the facilities.

II.C Unsupported Optional Facilities

Discuss the optional features of the service access protocol that are not implemented and discuss the general limitations of this adaptation.

II.D Additional Facilities

Discuss in detail any facilities not provided for or left undefined by the service access protocol.

II.E Host System Considerations

Discuss how the host SAPI must manipulate various host system primitives, system commands, user programs, etc. to perform its functions.

II.F Translation Considerations

II.F.1 Command Translation

Discuss the problems of translating command functions from those of the offloaded protocol into equivalent functions in the host system.

II.F.2 Data Translation

Discuss any data representation mismatch (see below), the translation problems involved, and how they are solved in this adaptation.

II.G References

Give references to all relevant documents.

Section III. Definition of the Adaptation

III.A Command Translation

Describe in detail the command translations that are performed by the host SAPI. For example, if the service defines a function called "Intersect", then this section is to give a detailed description of how the host SAPI performs this function.

III.B Data Translation

Describe in detail any data translations that are performed by the SAPIs. For example, if ASCII is converted to EBCDIC, the conversion table is to be given.

III.C Syntax

Describe the syntax of each service access protocol message, if it differs from the service access protocol specification. The description is to be in the same order and in the same form as in the service access protocol specification. It may be convenient to define additional facilities and fields at this time.

Data Representation Mismatch

The data representation mismatch problems which must be addressed by an adaptation description are:

a) character set mismatch and

b) data unit size mismatch.

The character sets and codes employed by the host and front end may differ. Any translation required is to be specified in the adaptation description.

The sizes of the data units employed by the host and the front end may differ. This may require redefinition of the syntax of service access protocol message fields in order to place them on convenient boundaries. This redefinition of fields may involve:

a) extending fields beyond the minimum lengths specified in the service access protocol specification, and/or

b) inserting padding fields where required.

Any transformation of data from one unit size to another is to be specified in the adaptation description.

BLANK PAGE

Host Front-End Protocol
Service Access Protocol Interpreter
State Table

This section contains the detailed state transition table for any SAPI interacting with a channel machine. This state table must be a subset of the state table for any SAPI. The SAPI and the channel machine communicate via channel interface events. The channel machine checks the events from the SAPI for consistency and rejects any inconsistent events.

Notation

States. All state names are printed with all capital letters. If the state name consists of two or more words, the words are separated by an underscore (_). Examples: NULL, SENDER_PENDING.

Events. All event names of events caused by a SAPI are either of the form:

s_<interface event name>_<event suffix>

where <interface event name> is one of the following:

begin
transmit
execute
end

and <event suffix> is either

c indicating a Command
or r indicating a Response

or of the form:

s_<interface event name>

where <interface event name> is one of the following:

ready
identify
status

All event names of events caused by a CPI or one of its channel machines are either of the form:

c_<interface event name>_<event suffix>

where <interface event name> is one of the following:

begin
transmit
execute
end

and <event suffix> is the same as above

or of the form:

c_<interface event name>

where <interface event name> is one of the following:

ready
status
accept
reject.

Event names are all lower case. Examples: s_begin_c, c_begin_c, s_ready, c_status.

A detailed description of each event can be found in the section entitled "Channel Interface".

Nomenclature

Acceptable/Unacceptable: an event may be found unacceptable by an SAPI if the SAPI detects an error in the Text field or if it is unable to fulfill the request described in the Text field due to lack of resources, insufficient access rights, etc.

Status ≠ 0: an event with this annotation indicates that the event to which it is a response was not successful.

Generic Requests: a process or user of an SAPI generates inputs to it. These inputs are modelled in the state table as "generic inputs". They are "request for service", "request termination", "send data", and "send Execute_Command". This specification does not define the parameters or any other characteristics of the interface between an SAPI and any higher level processes. It merely recognizes the existence of such interactions. This interface may be defined by the service level protocol or adaptation descriptions. It may be highly specific not only to

the SAPI but also to the environment in which it is implemented.

SAPI States

NULL: An SAPI in this state is not active.

PENDING: An SAPI in this state has requested its CPI to attempt to establish a virtual channel to its apposite. It has caused an s_begin_c event and is waiting for a c_begin_r event. (The channel machine has sent a Begin_Command and is waiting for a Begin_Response.)

TAKING BACK: An SAPI in this state has been requested to terminate communication with its apposite while it was in the PENDING state and has caused an s_end_c event before the c_begin_r event has come from the channel machine.

ESTABLISHED: An SAPI in this state has established communication with its apposite.

TERMINATING: An SAPI in this state has been requested to terminate communication with its apposite, and has caused an s_end_c event and is waiting for the c_end_r event acknowledging termination by its apposite.

SAPI States

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
NULL	request for service	PENDING	s_begin_c	
NULL	acceptable c_begin_c	ESTABLISHED	s_begin_r (Status=0)	set up service
NULL	unacceptable c_begin_c	- - - - -	s_begin_r (Status≠0)	
NULL	any other event	- - - - -	- - - - -	

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
PENDING	request termination	TAKING_BACK	s_end_c (flushing)	
PENDING	acceptable c_begin_r (Status=0)	ESTABLISHED	- - - - -	
PENDING	c_begin_r (Status≠0)	NULL	- - - - -	terminate service for user
PENDING	unacceptable c_begin_r (Status=0)	TERMINATING	s_end_c (flushing)	terminate service for user
PENDING	c_end_c (flushing or non-flushing)	NULL	s_end_r	terminate service for user
PENDING	c_reject	- - - - -	- - - - -	attempt error recovery
PENDING	any other event	- - - - -	- - - - -	

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
TAKING_BACK	c_begin_r (Status=0)	NULL	- - - - -	terminate service for user
TAKING_BACK	c_end_c (flushing)	NULL	- - - - -	terminate service for user
TAKING_BACK	c_end_r	NULL	- - - - -	terminate service for user
TAKING_BACK	c_reject	- - - - -	- - - - -	attempt error recovery
TAKING_BACK	any other event	- - - - -	- - - - -	

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
ESTABLISHED	request termination (flushing or non-flushing)	TERMINATING	s_end_c (flushing or non-flushing as requested)	
ESTABLISHED	send data	- - - - -	s_transmit_c	transfer data as allowed by c_ready
ESTABLISHED	send Execute_Command	- - - - -	s_execute_c	
ESTABLISHED	c_end_c (flushing or non-flushing)	NULL	s_end_r	terminate service for user
ESTABLISHED	c_execute_c	- - - - -	s_execute_r	
ESTABLISHED	c_execute_r	- - - - -	- - - - -	
ESTABLISHED	c_reject	- - - - -	- - - - -	attempt error recovery
ESTABLISHED	c_transmit_c	- - - - -	- - - - -	transfer data as allowed by s_ready
ESTABLISHED	any other event	- - - - -	- - - - -	

CURRENT STATE (SUB-STATE)	INPUT	NEXT STATE	OUTPUT	COMMENT
TERMINATING	c_end_c (flushing or non-flushing)	NULL	s_end_r	terminate service for user
TERMINATING	c_end_r	NULL	- - - - -	terminate service for user
TERMINATING	any other event	- - - - -	- - - - -	

HFP MAINTENANCE SERVICE

Introduction

The HFP Maintenance Service provides the management functions for the host to front end protocols. Since the present document fully specifies only the channel protocol, these management functions are here defined only in relation to the channel protocol. The HFP Maintenance Service may also provide management functions for the link protocol and the service access protocols. Whenever such additional management functions are defined for the HFP Maintenance Service, their definitions should be incorporated into the present specification.

The HFP Maintenance Service provides three management functions for the channel protocol. These are:

1) initializing host - front end communication,

2) recording CPI error reports, and

3) communicating CPI error reports between apposite CPIs.

The HFP Maintenance Service is implemented in both the host and in the front end. The two HFP Maintenance Service implementations communicate via an HFP Maintenance Service Access Protocol, which uses the channel protocol implementation as its communications medium.

HFP Maintenance Service
Service Protocol
Specification

I. HFP Maintenance Service Code Number

Ø (Zero)

II. Description of the HFP Maintenance Service

The HFP Maintenance Service provides management functions for the link, channel, and service access protocol interpreters (see Introduction, above).

Synopsis of Message Use

begin command

is sent by the host HFP Maintenance Service to initiate (or restart) host-front end communication.

begin response

is sent by the front end HFP Maintenance Service to confirm the establishment of communication.

end command

is sent by either the host or the front end HFP Maintenance Service to terminate channel level communication.

end response

is sent to confirm that channel level communication has been terminated.

execute command

not used

execute response

not used

transmit command

is sent by either the host or the front end HFP Maintenance Service to communicate error reports made by the CPIs.

III. Message Use

III A. begin command

III A 1. When sent

A begin_command is sent by the host HFP Maintenance Service when the host HFP implementation is ready to engage in channel level communication.

III A 2. Action on receipt

When the front end HFP Maintenance Service receives a begin_command, it determines whether or not the front end HFP implementation is prepared to engage in channel level communication, and, if so, it sends a begin_response.

III A 3. TEXT field syntax

empty

III A 4. TEXT field semantics

irrelevant

III B. begin response

III B 1. When sent

A begin_response is sent by the front end HFP Maintenance Service when the front end HFP implementation is prepared to engage in channel level communication.

III B 2. Action on receipt

After the host HFP Maintenance Service receives a begin_response, it proceeds to handle error reports from the CPI.

III B 3. TEXT field syntax

empty

III B 4. TEXT field semantics

irrelevant

III C. end command

III C 1. When sent

An end_command is sent by either HFP Maintenance Service to terminate channel level communication.

III C 2. Action on receipt

When the host or front end HFP Maintenance Service receives an end_command, it should notify all SAPI's using the HFP that service is ending, "clean up", and send an end_response.

III C 3. TEXT field syntax

empty

III C 4. TEXT field semantics

irrelevant

III D. end response

III D 1. When sent

An end response is sent to confirm that channel level communication has been terminated.

III D 2. Action on receipt

none

III D 3. TEXT field syntax

empty

III D 4. TEXT field semantics

irrelevant

III E. execute command

not used

III F. execute response

not used

III G. transmit command

III G 1. When sent

A transmit_command is sent by either EFP Maintenance Service to communicate an error report made by the channel protocol interpreter.

III G 2. Action on receipt

When an EFP Maintenance Service receives a transmit_command notifying it of an error, it should log the error and if possible attempt any error recovery.

III G 3. TEXT field syntax

Cause:	FIXED(8);
Header:	FIXED(72)
Diagnostics:	COMPLEX(?)

III G 4. TEXT field semantics

III G 4 (a). Cause

This field will contain one of the Status codes.

III G 4 (b). Header

This field contains the channel protocol HEADER for the message in which the error was detected.

III G 4 (c). Diagnostics

This field will contain any other diagnostic information that the CPI can provide relating to the error. If this transmit_command is reporting an error at the service access level, this field may contain the erroneous service access protocol message.

BLANK PAGE

Channel Protocol Response
Status Codes

Status	Meaning
Ø	Command was successful.
1	Channel non-existent: the Group and Member fields of a Command (other than a Begin_Command) referenced a channel unknown to the receiving CPI.
2	Illegal state: a message was received referencing a channel which was in a state for which the Command is an illegal input.
3	Command not implemented: a Command was received whose Type was legal but not implemented. Currently this can only be a Begin_Command or an Execute_Command.
5	Message too long: the number of bits in the Command exceeded the maximum permitted by the receiving CPI.
6	Service access protocol message error: an error in the service access protocol message contained in the TEXT field of the Command was detected by the SAPI.
7	Illegal Control field value: the Control field of the Command contained an undefined value.
32	Channel in use: the channel referenced in the Begin_Command was already assigned (i.e., not in the NULL state).
33	Service not implemented: the Service field in the Begin_Command specified an SAPI not implemented at the receiving site.
34	Insufficient resources: the receiver of the Begin_Command did not have sufficient resources for establishing the host to front end channel.
35	Out of sequence: a Transmit_Command was received and discarded whose Seq field was neither in sequence (equal to [<last received> + 1]) nor a duplicate (between [<last received> - 7] and <last received> inclusive) (see Flow Control).

36 Out of window: a Transmit_Command was received and discarded whose Seq field was between (<last received> + Credit + 1) and (<last received> + 8) inclusive (see Flow Control).

37 Bad channel polarity: the high-order bit of the Group field in the Begin_Command had the wrong value.

38 Service not operational: the Service field in the Begin_Command specified an SAPI which is implemented at the receiving site but which is temporarily unavailable.

39 Command discarded: the channel machine received a Transmit_Command or an Execute_Command, was in the SENDER_DRAINING or SENDER_TERMINATING state, and has discarded the Command without passing its TEXT field to the service access level.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: DTI-78012.C-INFE.14	2. GOVT ACCESSION NO.: AD-A100515
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): WWMCCS Host to Front End Protocols Specifications Version 1.0	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): John D. Day, Gary R. Grossman, Richard H. Howe	8. CONTRACT OR GRANT NUMBER(s): DCA100-77-C-0069
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Digital Technology Incorporated, 302 East John, Champaign, ILL 61820	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 32017K, 2782.00, 177
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Communications Agency, CCTC/C420, 11440 Isaac Newton Sq., N., Reston, VA 22090	12. REPORT DATE: 5 November 79
	13. NUMBER OF PAGES: 173
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release
Distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

No Restriction Distribution

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Network Front End
Host-to-Front End Protocol
Protocol Specification
COMMUNICATIONS NETWORKS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document presents the specifications of the WWMCCS Host to Front End protocols. A brief overview of the WWMCCS Network Front End Protocol architecture is presented. The link protocol is functionally specified. The "Channel" protocol is completely specified. The complete channel protocol specification includes a narrative overview of the channel protocol mechanisms, a detailed treatment of each channel protocol message and event type, and a complete state table. A meta-specification for the service access protocols is presented.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)